Metropolia Ammattikorkeakoulu
Mediatekniikan koulutusohjelma

Mia Santanen

Pienoisohjelman rajapintojen kääntäminen toiselle pienoisohjelma-alustalle

Insinöörityö 4.5.2010

Ohjaaja: yliopettaja Kari Salo
Ohjaava opettaja: yliopettaja Petri Vesikivi

Metropolia Ammattikorkeakoulu **Insinöörityön tiivistelmä**

Tekijä Otsikko Sivumäärä Aika	Mia Santanen Pienoisohjelman rajapintojen kääntäminen toiselle pienoisohjelma-alustalle 68 sivua 4.5.2010
Koulutusohjelma	mediatekniikka
Tutkinto	insinööri (AMK)
Ohjaaja Ohjaava opettaja	yliopettaja Kari Salo yliopettaja Petri Vesikivi

Pienoisohjelmat (engl. widget) on suunniteltu suorittamaan yhtä tehtävää. Tehtävät voivat olla hyödyllisiä, kuten uusimpien uutisten hakemista, tai viihteellisiä, kuten pelejä. Niiden tärkeimmät ominaisuudet ovat pienikokoisuus ja siirreltävyys, jotka mahdollistavat pienoisohjelmien käytön esimerkiksi tietokoneen työpöydällä tai matkapuhelimessa. Pienoisohjelmat käyttävät yleisiä www-teknologioita, jolloin kuka tahansa, joka osaa tehdä Internet-sivuja, osaa myös tehdä mobiilisovelluksia.

Insinöörityön tavoitteena oli kehittää menetelmä, jolla pienoisohjelmaa voitaisiin käyttää toisella pienoisohjelma-alustalla. Standardoitua tapaa kehittää pienoisohjelmia ei ole, joten eri teknologioita ja rajapintoja käyttäviä laitteita ja pienoisohjelmia on useita. Insinöörityön avulla pyrittiin parantamaan yhteensopivuutta kahden eri pienoisohjelma-alustan välillä.

Menetelmänä päädyttiin käyttämään JavaScript-käärekirjastoa, jolla luotiin alkuperäisen pienoisohjelma-alustan objekteja ja -metodeita vastaava rakenne kohdealustan toiminnallisuudella. Työn tuloksena syntyi JavaScript-käärekirjasto, esimerkkisovellukset, dokumentaatiot ja automatisoitu alustustiedostojen kääntäjä. Käärekirjasto sisälsi kaksi käännettyä rajapintaa, ja se julkaistiin avoimena lähdekoodina, jota muut pienoisohjelmien kehittäjät voisivat tulevaisuudessa laajentaa. Esimerkkisovellukset kattoivat kaksi alkuperäisen pienoisohjelma-alustan rajapintoja käyttävää pienoisohjelmaa ja yhden perinteistä JavaScriptiä käyttävän pienoisohjelman. Dokumentaatiot sisälsivät yksityiskohtaiset ohjeet pienoisohjelman kääntämiseen ja yleiset ohjeet käärekirjaston laajentamiseen.

Alustustiedostojen kääntäjä nopeutti pienoisohjelmien kääntämisprosessia entisestään automatisoimalla alustustiedostojen muunnokset useiden pienoisohjelma-alustojen alustustiedostojen kesken.

Kääntämättömiä rajapintoja on jäljellä useita. Työn tuloksena syntynyttä käärekirjastoa voisi tulevaisuudessa laajentaa kattamaan muita rajapintoja ja mahdollisesti jopa muidenkin ympäristöjen rajapintoja. Käärekirjastosta voisi olla apua, kunnes mobiiliteollisuuden tahot sopivat yhteisestä standardista, joilla pienoisohjelmia kehitetään.

Hakusanat	bondi, wrt, widget, pienoisohjelma, pienohjelma

Helsinki Metropolia University of Applied Sciences **Abstract**

Author Title Number of Pages Date	Mia Santanen Porting widget API to another widget user agent 68 4 May 2010
Degree Programme	Media Technology
Degree	Bachelor of Engineering
Instructor Supervisor	Kari Salo, Principal Lecturer Petri Vesikivi, Principal Lecturer

Widgets are small applications that are designed to perform one task. The tasks can vary from retrieving the latest news to entertainment purposes. The most important properties of a widget are size and portability, which allow the widget to be used in different environments ranging from computer desktops to mobile phones. Widgets use common web technologies, so anyone who can create a website can also create a mobile widget.

The aim of this thesis was to develop a method for porting widgets to another widget user agent. There are a variety of technologies and application programming interfaces used in widget development because there is no standardized way to develop widgets. This thesis improved the compatibility between two widget user agents.

This thesis explored the differences between the two platforms and resulted in a JavaScript wrapper library, sample widgets, documentation and an automated configuration file converter. The wrapper library consisted of two ported application programming interfaces (APIs) and was published as open source. Open source enabled other widget developers to extend the wrapper library in the future. Sample widgets consisted of two widgets that use the original widget user agent APIs and one that uses plain JavaScript. Documentations included a step by step guide for porting widgets and a general guide for extending the wrapper library. Configuration file converter simplified the widget porting process by automating the configuration file conversions between multiple widget user agents.

There are a number of APIs left unported. In the future, the wrapper library could be extended to include other APIs or even APIs from another platform. The wrapper library could reduce the gap between different platforms until the mobile industry agreed on a standardized widget development method.

Keywords	bondi, wrt, widget, api, porting

Sisällys

Lyhenteet, käsitteet ja määritteet

Ajax	Asynchronous JavaScript and XML. Joukko menetelmiä, joiden avulla verkkosovellukset voivat siirtää tietoa selaimen ja palvelimen välillä päivittämättä verkkosivua.
Callback	Callback on viittaus koodiin, joka suoritetaan, kun sitä kutsuvan koodin tietyt vaatimukset täyttyvät.
CSS	Cascading Style Sheets. Tyylikieli, jolla määritellään muun muassa verkkosivujen ulkoasu ja muotoilu. Sen tarkoituksena on erottaa muotoilu sisällöstä.
DOM	Document Object Model. Alustasta ja ohjelmointikielestä riippumaton rajapinta, joka mahdollistaa dokumenttien sisällön, rakenteen ja tyylin dynaamisen käytön ja päivittämisen suoraan ohjelmasta tai komentosarjasta.
ECMAScript	Komentosarjakieli, joka perustuu muun muassa Netscapen JavaScriptiin ja Microsoftin JScriptiin. Ecma International standardisoi kielen määritelmään ECMA-262.
JavaScript	Asiakaspuolen ohjelmoinnissa yleisimmin käytetty komentosarjakieli. Standardoitua JavaScriptiä nimitetään usein ECMAScriptiksi.
JSON	JavaScript Object Notation. Kevyt tiedonvaihtomuoto, joka on ohjelmointikielestä riippumaton. JSON perustuu ECMAScriptiin.
Komentosarjakieli	Ohjelmointikieli, jossa komennot tulkataan ja suoritetaan yksitellen. Ohjelmaa ei tarvitse kääntää.

Internet-mediatyyppi	Internet-standardi, joka määrittelee sisällön tiedostomuodon. Alun perin kutsuttu MIME-tyypiksi.
Pienoisohjelma-alusta	Ohjelmisto, joka mahdollistaa pienoisohjelmien suorittamisen tietyissä ympäristöissä, kuten työpöydällä tai matkapuhelimessa.
SDK	Software Development Kit. Ohjelmisto, jonka avulla voi kehittää sovelluksia tietylle ohjelmistoalustalle.
URL	Uniform Resource Locator. Merkkijono, jolla määritellään tiedon paikka. Yleisimmin käytetty verkkosivujen osoitteissa.
W3C	World Wide Web Consortium. Yhteisö, joka kehittää WWW-standardeja.
WRT	Web Runtime. Nokian kehittämä pienoisohjelma-alusta.
XHTML	Extensible Hypertext Markup Language. Verkkosivujen merkintäkieli, joka täyttää XML:n muotovaatimukset.
XML	Extensible Markup Language. XML kuvaa luokkaa, joka sisältää XML-dokumenteiksi kutsuttuja objekteja ja osittain niitä käsittelevien ohjelmien toimintaa.
ZIP	Tiedon pakkaus- ja arkistointimuoto.

1 Johdanto

Viime vuosina mobiiliverkon teknologiat ja tuotteet ovat kehittyneet nopeasti. Yhtenäistettyä tapaa kehittää verkko- ja mobiilisovelluksia ei kuitenkaan ole, mikä aiheuttaa sen, että eri yrityksillä ja yhteisöillä on omat standardinsa sovelluksille. Tämä taas johtaa siihen, että sovellukset on yleensä suunniteltu yhdelle alustalle tai jopa yhdelle tietylle laitteelle. Mobiiliverkon tulevaisuuden kehityssuunta ja menestys voivat kärsiä ilman mobiiliteollisuuden arvoketjun jäsenten yhteistä panostusta standardisoida lähestymistapaa, jolla sovellus käyttää mobiililaitteen paikallista dataa. Laitteesta ja alustasta riippumattomia sovelluksia ei kehitetä, jos sovellusten täytyy käyttää erilaisia ohjelmointirajapintoja eri laitteilla ja alustoilla suorittaakseen saman tehtävän. [1.]

Yksi tällainen sovellusryhmä ovat pienoisohjelmat (engl. widget). Pienoisohjelma on yleinen termi sovellukselle, joka on suunniteltu suorittamaan yhtä tehtävää. Pienoisohjelmat voidaan luokitella niiden käyttöympäristön mukaan työpöytäpienoisohjelmiin, selainpienoisohjelmiin, GUI-pienoisohjelmiin ja mobiilipienoisohjelmiin. Työpöytä- ja mobiilipienoisohjelmat vaativat pienoisohjelma-alustaksi kutsutun ohjelmiston toimiakseen. [2.]

Tämä insinöörityö keskittyy mobiilipienoisohjelmiin ja kattaa niistä vain BONDI- ja WRT-ympäristöt sekä niille tarkoitetut pienoisohjelmat. BONDI on nouseva mobiiliteollisuuden hanke, jonka tarkoituksena on standardisoida rajapintoja, joilla pienoisohjelmat voivat käyttää laitteen paikallista dataa tehokkaasti ja turvallisesti. Insinöörityön tekovaiheessa BONDI-hanke oli alfaversiossa 1.0, josta se on vuoden 2010 alussa edennyt jo versioon 1.1. WRT (Web Runtime) on Nokian matkapuhelimissa käytetty pienoisohjelma-alusta.

Insinöörityön tavoitteena on tutkia ja toteuttaa toimintatapa, jolla BONDI-pienoisohjelmaa voidaan käyttää Nokian WRT-alustalla. Työ tehdään Forum Nokialle, joka on Nokian maailmanlaajuinen kehittäjätukiorganisaatio. Forum Nokia tarjoaa kehittäjille muun muassa pääsyn Nokian laitteisiin testausta ja kehitystä varten, uusimpia mobiiliteollisuuden uutisia ja teknistä tietoa Nokian laitteista ja

mobiilisovellusten kehitysalustoista. Nokia tekee yhteistyötä BONDI-hankkeen kanssa, minkä takia on ajankohtaista parantaa yhteensopivuutta näiden kahden alustan välillä.

2 Pienoisohjelmat

2.1 Pienoisohjelmien määritelmä

Pienoisohjelmat (engl. widget) ovat sovelluksia, joiden perimmäinen tarkoitus on käytännöllisyys. Ne yhdistävät käyttäjän ensisijaiseen tietolähteeseen tarjoten esimerkiksi uusimpia uutisia, säätiedotuksia tai oikopolun usein käytettyihin työkaluihin. Pienoisohjelmien tärkeimpiä ominaisuuksia ovat pienikokoisuus ja siirrettävyys: käyttäjä voi lisätä pienoisohjelman tietokoneensa työpöydälle, verkkosivuilleen tai mobiililaitteeseensa [3].

Sanaa *widget* käytettiin kuvaamaan käyttöliittymän graafista elementtiä ensimmäisen kerran vuonna 1988 Athena-projektissa, joka on Massachusetts Institute of Technologyn (MIT), Digital Equipment Corporationin ja IBM:n kehittämä akateeminen tietojenkäsittely-ympäristö. Widget-sanaan päädyttiin, koska se liittyi projektissa käytettyyn X Window -ohjelmointirajapintakirjastoon, ja niiden yhteiset alkukirjaimet koettiin hyödyllisiksi. Lisäksi muilla mahdollisilla vastaavilla termeillä oli sopimattomia mielleyhtymiä. [4.]

Nykyään sanaa *widget* käytetään yleisesti kuvaamaan kaikkia pienoisohjelmia, jotka yleensä jaetaan neljään luokkaan niiden käyttöympäristön mukaan: selainpienoisohjelmiin, työpöytäpienoisohjelmiin, GUI-pienoisohjelmiin ja mobilipienoisohjelmiin.

2.2 Selainpienoisohjelmat

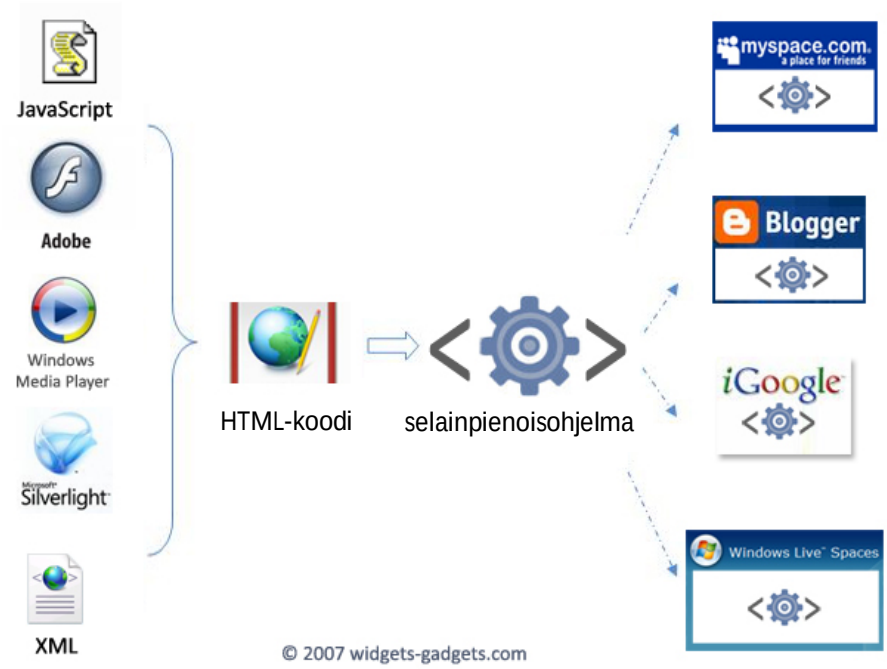
Selainpienoisohjelmat ovat koodinpätkiä, jotka lisätään verkkosivulle joko deklaratiivisesti tai dynaamisesti. Toisin kuin muut pienoisohjelmat, selainpienoisohjelmat sijaitsevat palvelimella ja ne lisätään verkkosivuille, ennen kuin

sivut näytetään käyttäjälle. Selainpienoisohjelmat tarjoavat pitkälti samat toiminnallisuudet kuin muut pienoisohjelmat, mutta ne eroavat pakkaustavassa, turvallisuusmallissa ja ohjelmointirajapinnoissa.

Selainpienoisohjelmia ei yleensä pakata eikä toimiteta yhtenä tiedostona, vaan ne käynnistetään ECMAScriptin, HTML:n ja CSS:n avulla. Jotkin selainpienoisohjelmat käyttävät myös alustustiedostoja, kuten tavalliset pienoisohjelmat. [2.]

Koska selainpienoisohjelmat ovat osa verkkosivua, ne ovat sidottuja selaimen asettamiin turvallisuusrajoituksiin eivätkä täten voi käyttää samanlaisia ohjelmointirajapintoja kuin muut pienoisohjelmat. Tämä tarkoittaa sitä, että ne eivät voi suorittaa verkko-osoitteiden välisiä pyyntöjä (engl. cross-domain requests), päästä käsiksi laitteen resursseihin itsenäisesti tai suorittaa järjestelmätason komentoja, kuten luoda tai poistaa tiedostoja käyttäjän laitteesta. [2.]

Kuvassa 1 esitetään selainpienoisohjelman rakenne ja käyttö. Kuvan vasemmassa reunassa oleva teknologioiden joukko sisältää esimerkkejä selainpienoisohjelmien mahdollisista teknologioista ja mediatyypeistä. Jotta selainpienoisohjelman sisältö voidaan näyttää verkkosivulla, se muotoillaan HTML- tai XHTML-esitystavan mukaiseksi, ja jotta koodinpätkästä tulisi selainpienoisohjelma, sen täytyy olla uudelleenkäytettävä ja itsenäisesti toimiva. Tämän jälkeen selainpienoisohjelma voidaan lisätä mille tahansa verkkosivulle, joista on lueteltu muutama esimerkki kuvan 1 oikeassa reunassa.



Kuva 1: Selainpienoisohjelman rakenne ja käyttö [5].

2.3 Työpöytäpienoisohjelmat

Työpöytäpienoisohjelmat tunnetaan myös nimillä gadgetit, deskletit, screenletit ja plasmoidit. Niille kaikille on ominaista se, että ne sijaitsevat tietokoneen työpöydällä. Eri käyttöjärjestelmät ja sovellukset käyttävät työpöytäpienoisohjelmista eri nimityksiä. Työpöytäpienoisohjelmat vaativat pienoisohjelma-alustaksi kutsutun ohjelmiston toimiakseen. Jotkin käyttöjärjestelmät sisältävät jo pienoisohjelma-alustan, jolloin erillistä ohjelmistoa ei tarvitse ladata käyttääkseen pienoisohjelmia. Tällaisia käyttöjärjestelmiä ovat esimerkiksi Windows 7, jonka pienoisohjelma-alustan nimi on Windows Desktop Gadgets, ja Mac OS X, jonka pienoisohjelma-alusta on Apple Dashboard. Esimerkkejä erikseen asennettavista pienoisohjelma-alustoista ovat Google Desktop Sidebar, Opera ja Yahoo! Widgets Engine (aiemmalta nimeltään Konfabulator) [6].

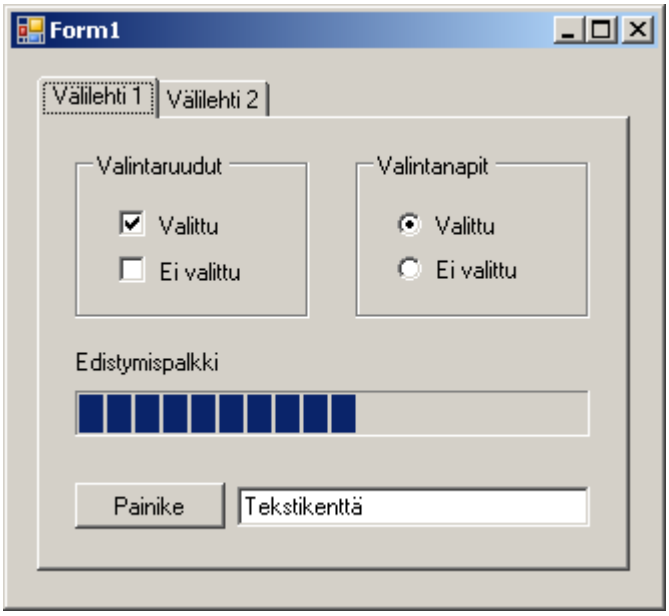
Kuvassa 2 on esimerkki Apple Mac OS X -käyttöjärjestelmän työpöydästä, jossa on runsaasti erilaisia pienoisohjelmia. Useimmissa työpöytien pienoisohjelma-alustoissa pienoisohjelmien paikkaa voi vapaasti siirrellä työpöydän reunojen sisällä. Poikkeuksen

tekee Windows Vistan Windows Sidebar, jossa työpöytäpienoisohjelmia voi lisätä ainoastaan ruudun reunassa sijaitsevaan sivupalkkiin.

Kuva 2: Apple Dashboard -pienoisohjelmia työpöydällä [7].

2.4 GUI-pienoisohjelmat

GUI-pienoisohjelmat ovat tietokoneohjelman graafisen käyttöliittymän (graphical user interface, GUI) elementtejä, joiden avulla käyttäjä pystyy muuttamaan tietoa [8]. Tällaisia elementtejä ovat esimerkiksi kuvassa 3 havainnollistetut tekstikentät, painikkeet, valintaruudut ja välilehtivalikot. GUI-pienoisohjelmat eroavat muista pienoisohjelmista, koska termillä tarkoitetaan yksittäisiä komponentteja kokonaisen sovelluksen sijaan. Jos kuvan 3 sovellus muutettaisiin esimerkiksi työpöytäpienoisohjelmaksi, se olisi yksi kokonainen pienoisohjelma eikä joukko komponentteja.



Kuva 3: Esimerkkejä GUI-pienoisohjelmista.

2.5 Mobiilipienoisohjelmat

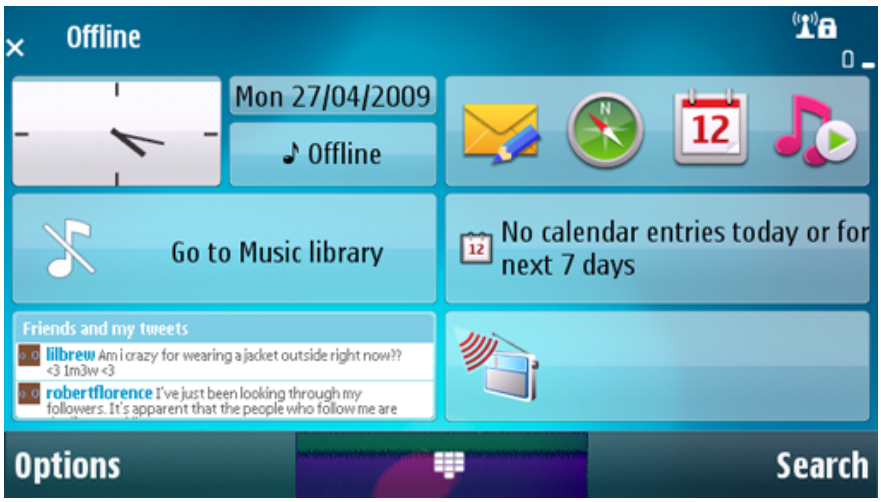
Mobiilipienoisohjelmat ovat matkapuhelimessa tai muussa mobiililaitteessa toimivia sovelluksia. Mobiilipienoisohjelmien tarkoitus on helpottaa laitteen käyttöä, näyttää käyttäjälle yksilöityä tietoa tai esimerkiksi viihdyttää pelin avulla [9]. Mobiilipienoisohjelmat ovat samankaltaisia ja käyttävät samoja WWW-teknologioita kuin työpöytäpienoisohjelmat. Ne eroavat toisistaan käyttöympäristössä ja pienoisohjelma-alustan rajapinnoissa. Mobiilipienoisohjelmien toimintaan tarvittava pienoisohjelma-alusta on joko erikseen asennettava sovellus, kuten BONDI-hankkeessa, tai mobiililaitteen käyttöjärjestelmään sisällytetty ohjelmisto, kuten Nokian matkapuhelimissa oleva Web Runtime (WRT).

Mobiilipienoisohjelmat voivat olla koko sivun täyttäviä, tavallisen applikaation näköisiä sovelluksia, tai pieniä ruudulla liikuteltavia elementtejä. Jotkut pienoisohjelma-alustat tukevat pienoisohjelman kahta esitystapaa, jolloin sama pienoisohjelma voi olla sekä kompakti, pieni elementti ruudulla että koko ruudun kokoinen enemmän informaatiota sisältävä sovellus. Kuva 4 on esimerkki koko ruudun täyttävästä pienoisohjelmasta, jolla haetaan tietoa Twitteristä, suositusta mikrobloggaussivustosta.

Esimerkkipienoisohjelmassa on useita eri toimintoja, joille on oma näkymänsä. Näkymät erottelevat esimerkiksi hakutulokset ja tallennetut suosikit.

Kuva 4: Esimerkki koko sivun täyttävästä mobiilipienoisohjelmasta [10].

Kuvassa 5 on esitelty Nokia N97 -matkapuhelinmallin aloitusnäyttö (engl. home screen). Aloitusnäyttö koostuu useasta pienoisohjelmasta, jotka ovat poistettavissa ja siirreltävissä. Osa aloitusnäytön pienoisohjelmista tulee käyttöjärjestelmän mukana, mutta siihen voi myös lisätä omia pienoisohjelmia.



Kuva 5: Esimerkki pienistä, ruudulla liikuteltavista pienoisohjelmista [11].

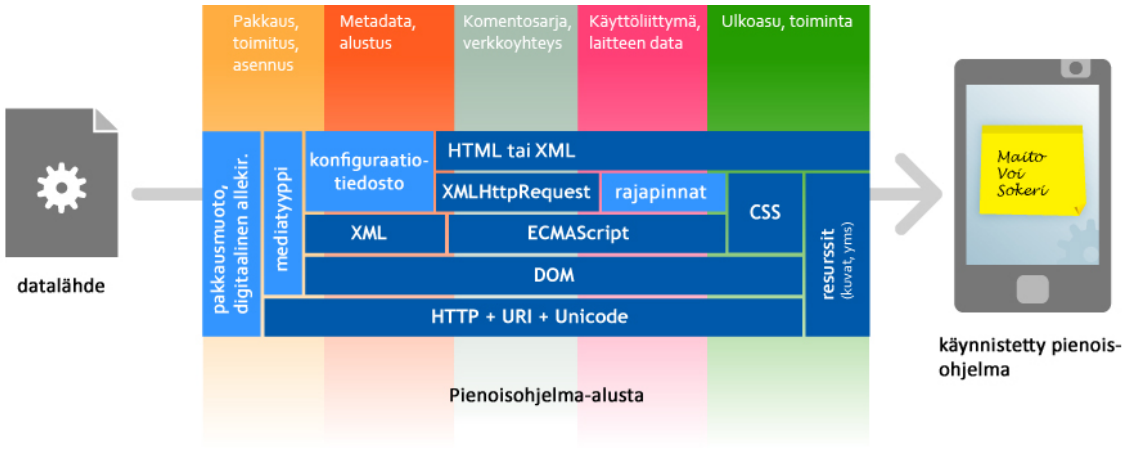
2.6 Pienoisohjelmien arkkitehtuuri

Pienoisohjelmat käyttävät yleisiä www-teknologioita, kuten HTML:ää, CSS:ää ja JavaScriptia. Pienoisohjelma koostuu alustustiedostosta, yhdestä tai useammasta HTML-tiedostosta ja mahdollisista muista resursseista, kuten CSS-tyylitiedostoista,

JavaScript-tiedostoista tai kuvista, jotka on pakattu *zip*-arkistoksi ja nimetty uudelleen pienoisohjelma-alustan spesifikaation mukaisesti. Kuvassa 6 on esimerkki pienoisohjelman tiedostoista ja niiden sisällöistä. Kuva 6 jaetaan neljään osaan, joista vasen yläkulma kuvaa *index.html*-tiedoston sisältöä, oikea yläkulma *widget.js*-tiedoston sisältöä, vasen alakulma alustustiedostoa *config.xml* ja oikea alakulma käyttäjälle esitettyä näkymää.

Kuva 6: Esimerkki pienoisohjelman sisällöstä.

Kuvassa 7 on esitetty yleinen pienoisohjelman teknologiarakenne. Se on suuntaa antava eikä edusta minkään yksittäisen pienoisohjelma-alustan teknologiarakennetta [2]. Kuvassa alimpana on pienoisohjelma-alusta, joka tunnetaan myös nimillä *widget engine* ja *host runtime*. Se on ohjelmisto, joka mahdollistaa pienoisohjelman toiminnan ja tarjoaa rajapinnan pienoisohjelman ja laitteen väliselle tiedonvaihdolle. Vasemmalla on pienoisohjelman datalähde, joka voi olla laitteessa sijaitsevaa tai Internetin välityksellä haettavaa dataa.



Kuva 7: Tyypillinen pienoisohjelman teknologiarakenne [2].

Pienoisohjelman toiminnallisuus jaetaan viiteen luokkaan, jotka ovat kuvassa 7 ylimpien laatikoiden sisällä. Ensimmäinen luokka on pakkaus, toimitus ja asennus, johon kuuluu pienoisohjelman pakkausmuoto, digitaalinen allekirjoitus ja mediatyyppi. Pakkausmuoto ja mediatyyppi määrittelevät, mille pienoisohjelma-alustalle pienoisohjelmat ovat tarkoitettuja, ja digitaalinen allekirjoitus varmentaa pienoisohjelman sisällön ja toimittajan.

Toinen luokka on metadata ja alustus, johon kuuluu alustustiedosto, joka sisältää tarvittavat asetukset pienoisohjelman toiminnalliselta kannalta ja mahdollisesti lisätietoa pienoisohjelmasta ja sen tekijästä. Alustustiedosto on XML-muotoinen.

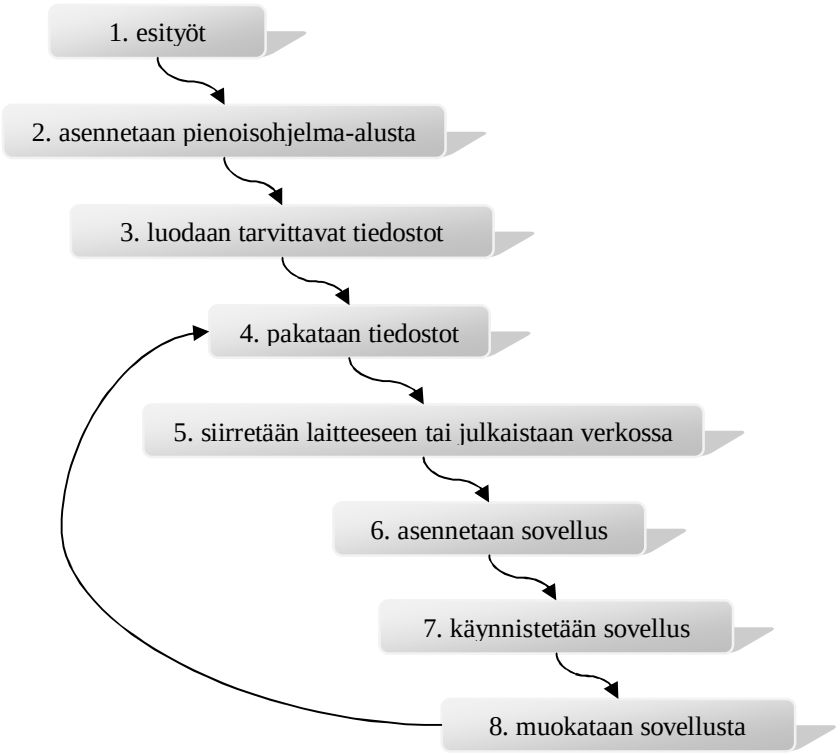
Kolmas luokka on komentosarja ja verkkoyhteys, jolla toteutetaan pienoisohjelman toiminnallisuus ja yhteys verkkoon. Luokka sisältää yleisiä www-teknologioita, kuten HTML, XML, XMLHttpRequest, ECMAScript, DOM, HTTP, URI ja Unicode.

Neljäs luokka on käyttöliittymä ja pääsy laitteen dataan. Käyttöliittymä luodaan samoilla www-teknologioilla kuin kolmannessa luokassa. Pienoisohjelma saattaa käyttää laitekohtaista dataa, kuten matkapuhelimen yhteystietoja tai tietokoneen suorittimen kuormitustasoa. Laitekohtainen data on saatavilla pienoisohjelma-alustan ohjelmointirajapintojen avulla.

Viides luokka on ulkoasu ja toiminto. Aiemmin luotu käyttöliittymä muotoillaan CSS-tyylikielellä. Pienoisohjelmissa voi käyttää myös ulkoasuun tai toimintoihin liittyviä resursseja, kuten kuvia, videota ja ääntä. Pienoisohjelma-alusta sisältää menetelmän, joilla nämä resurssit tulkitaan. Kuvan 7 oikealla puolella on esimerkki siitä, miltä pienoisohjelma voisi näyttää loppukäyttäjälle. Käyttäjän ei tarvitse tuntea pienoisohjelman teknologioita käyttääkseen sitä.

2.7 Kehitys ja käyttöönotto

Pienoisohjelmien kehitys ja käyttöönotto seuraa yleensä kuvassa 8 esitettyjä vaiheita. Ensimmäiseen vaiheeseen kuuluu esityö, joka sisältää taustatietojen selvittelyä, kuten dokumentaatioon ja ympäristöön tutustumista ja sovelluksen suunnittelua. Toinen vaihe on valinnainen; mikäli pienoisohjelma-alusta ei sisälly laitteen käyttöjärjestelmään, se tulee asentaa. Kolmannessa vaiheessa luodaan tarvittavat tiedostot, jotka pienoisohjelmien tapauksessa ovat alustustiedosto, HTML-tiedosto ja mahdolliset muut resurssit. Kun kaikki tarvittavat tiedostot on luotu, ne pakataan neljännessä vaiheessa pienoisohjelmien pakkausspesifikaation mukaisesti, yleensä uudelleennimetyiksi *zip*-arkistoiksi.



Kuva 8: Pienoisohjelmien kehitysprosessi [12, s. 53].

Viidennessä vaiheessa pienoisohjelma siirretään laitteeseen tai julkaistaan verkossa. Mikäli pienoisohjelman julkaisee verkossa, sille tulee asettaa asianmukainen Internet-mediatyyppi. Laitteen sijaan pienoisohjelman voi siirtää myös emulaattorin tiedostojärjestelmään. Kuudennessa vaiheessa sovellus asennetaan laitteeseen tai emulaattoriin avaamalla se laitteen tai emulaattorin resurssienhallintaohjelmalla, jolloin järjestelmä hoitaa asentamisen. Asennuksen onnistuttua pienoisohjelma lisätään automaattisesti laitteen sovelluslistaan, jolloin sen voi käynnistää samalla tavalla kuin minkä tahansa muun sovelluksen.

Seitsemännessä vaiheessa pienoisohjelma käynnistetään ja testataan. Mikäli siitä löytyy parantamisen varaa, muokataan pienoisohjelman tiedostoja vaiheessa 8. Muokkaamisen jälkeen täytyy palata vaiheeseen 4 ja pakata tiedostot uudelleen. Vaiheita 4–8 toistetaan, kunnes pienoisohjelma vastaa suunnitelmia.

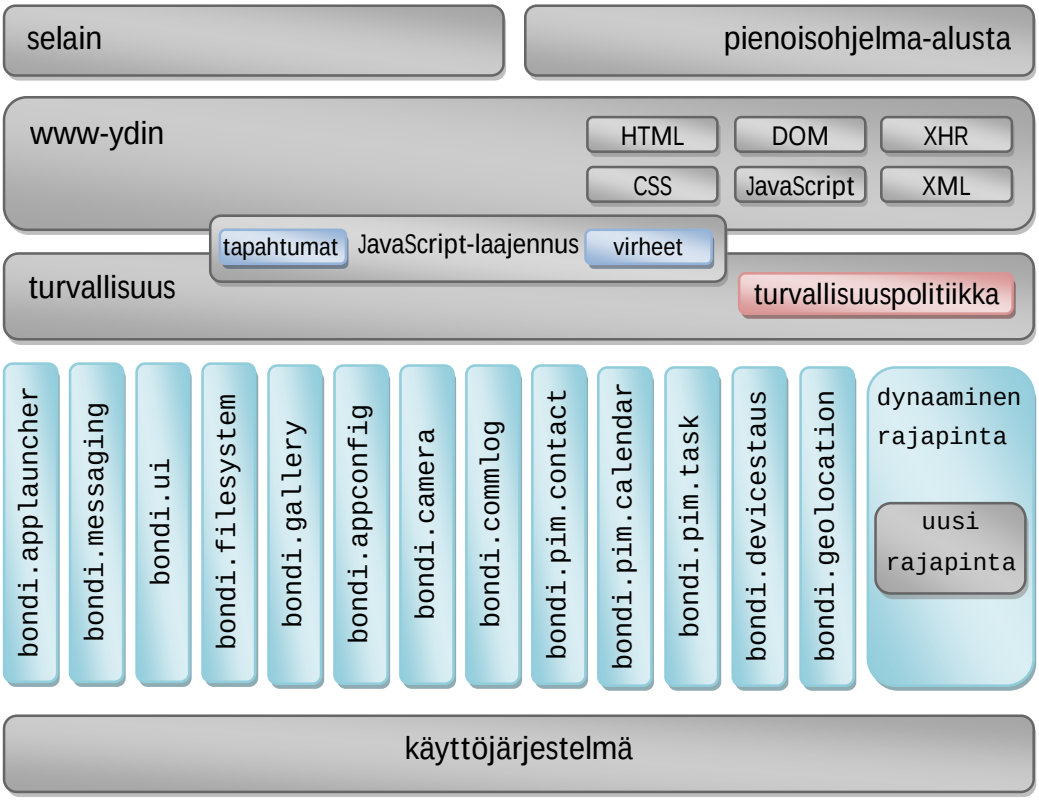
3 BONDI-hanke

3.1 BONDI-arkkitehtuuri

Open Mobile Terminal Platform (OMTP) aloitti BONDI-hankkeen edistääkseen mobiiliteollisuuden laitteesta ja alustasta riippumattomien sovellusten kehitystä. Hankkeen tavoitteena on standardisoida joukko rajapintoja, joilla sovellukset käyttävät laitteen paikallista dataa. Rajapintojen lisäksi BONDI tarjoaa helposti ymmärrettävän ja käyttäjän hallinnassa olevan turvallisuuskäytännön, jolla suojataan käyttäjän laitetta ja käyttäjän tietoja. [1.]

Open Mobile Terminal Platform (OMTP) on kahdeksan teleoperaattorin perustama voittoa tavoittelematon mobiiliteollisuuden järjestö. Järjestön tavoitteena on parantaa mobiililaitteiden turvallisuutta ja helpottaa matkaviestinnän datapalvelujen käyttöä. OMTP on kasvanut perustajajäsenten operaattoritaustasta kattamaan koko mobiiliteollisuuden alan, minkä seurauksena järjestö sisältää nykyään noin 35 jäsentä ja sponsoria. Jäsenet ovat operaattoreiden lisäksi matkapuhelinvalmistajia, sirunvalmistajia ja ohjelmistojen ja käyttöjärjestelmien kehittäjiä. [13.]

Kuvassa 9 on esitetty BONDI-pienoisohjelman arkkitehtuuri. Alimmassa kerroksessa on mobiililaitteen käyttöjärjestelmä. Vuoden 2009 lopussa BONDI tuki vain Windows Mobile -käyttöjärjestelmää. Käyttöjärjestelmän päällä on BONDI-ohjelmointirajapintojen kerros, jolla hallitaan pääsyä laitteen dataan. Rajapintojen käyttöä taas hallitsee turvallisuuskäytäntöjen kerros, joka määrittelee, mihin rajapintaan sovelluksella on oikeus päästä. JavaScript-laajennus liittyy oleellisesti sekä turvallisuuskäytännön kerrokseen että sen yläpuolella olevaan www-ytimeen. JavaScript-laajennus määrittelee tapahtumat ja virheilmoitukset, joiden avulla eri kerrokset viestivät keskenään.



Kuva 9: BONDI-arkkitehtuuri [15].

Www-ydin vastaa XHTML-, SVG-, CSS- ja JavaScript-teknologioiden käsittelystä ja grafiikan esittämisestä. Sekä pienoisohjelma-alusta että selain toimivat www-ytimen päällä. Pienoisohjelma-alusta on kokoelma kaikista komponenteista, joita tarvitaan pienoisohjelmien käyttöön [14, s. 10]. Pienoisohjelma-alustan odotetaan vastaavan pienoisohjelmien asennuksesta ja poistosta. Selain on ohjelmisto, jonka avulla selataan verkkosivuja. Verkkosivujen näyttämisen lisäksi selaimissa on monia muita toimintoja, kuten kirjanmerkit ja välimuisti. Kuvassa 9 ylimpänä olevien selaimen ja pienoisohjelma-alustan lopputuotteita ovat verkkosivu ja pienoisohjelma, joiden toimintaan tarvitaan arkkitehtuurissa alempana olevia elementtejä.

3.2 BONDI-ohjelmointirajapinnat

Yleistä BONDI-ohjelmointirajapinnoista

BONDIn päätavoitteena on tarjota yhteiset ohjelmointirajapinnat kehittäjille, jotta heidän ei tarvitse käyttää erilaisia rajapintoja suorittaakseen samaa tehtävää eri laitteilla tai alustoilla [16]. BONDI-rajapintoja käytetään JavaScriptin avulla. Rajapinnan täytyy olla sallittu ensin pienoisohjelman alustustiedostossa, jotta sitä voisi käyttää. Rajapintojen käyttö sallitaan alustustiedoston `feature`-elementin avulla. BONDIn alfaversiossa 1.0 ohjelmointirajapinnat on jaettu 14 moduuliin, jotka koostuvat toisiinsa liittyvien rajapintojen joukosta.

Tässä insinöörityöraportissa käsitellään tarkemmin mobiililaitteen tilamoduulia (`bondi.devicestatus`) ja paikantamismoduulia (`bondi.geolocation`), koska ne ovat oleellisia työn kannalta. Muut moduulit käydään yleisellä tasolla läpi.

Mobiililaitteen tilamoduuli

Tiedot mobiililaitteen tilasta sijaitsevat ryhmitellyssä puurakenteessa. Laitteen tilarajapinta määrittelee menetelmiä, joilla voi selata puurakennetta, hakea tai määrittää valitun ominaisuuden arvo ja luoda asynkronisia ilmoituksia, kun joku arvo muuttuu. Mobiililaitteen tilamoduulin käyttö sallitaan pienoisohjelman alustustiedoston `feature`-elementissä nimellä `http://bondi.omtp.org/api/devicestatus`. [17.]

Laitteen tilatiedot jaetaan neljään ryhmään, jotka ovat sanasto (vocabulary), aspekti (aspect), komponentti (component) ja ominaisuus (property). Sanastot ovat nimiavaruuksia, jotka koostuvat eri aspekteista. Aspektit keräävät tietoa, joka liittyy johonkin laitteen osaan. Aspekteihin liittyvä tieto on jaettu ominaisuuksien mukaan. Aspekti voi sisältää useita komponentteja, joihin ominaisuudet kohdistuvat ja jotka on tarkoitettu kuvaamaan saman aspektin erilaisia tapauksia. [17.]

Taulukossa 1 on esitelty kaksi aspektia ja niiden mahdollisia komponentteja ja ominaisuuksia. Ominaisuudet ovat samat jokaiselle aspektin komponentille. Esimerkiksi mobiililaitteen akun jäljellä olevan varaustason voi hakea erikseen sisäänrakennetulle akulle (ensisijainen) ja lisäakulle (toissijainen).

Taulukko 1: Esimerkki laitteen tilatiedoista.

Sanasto	Aspekti	Komponentit	Ominaisuudet
BONDI	muisti	fyysinen muisti, virtuaalinen muisti	yhteensä, käytettävissä
BONDI	akku	ensisijainen akku, toissijainen akku	varaustaso, kapasiteetti, lataustila

Laitteen tilarajapinta määrittelee neljä metodijoukkoa, joilla tilatietoja voidaan käyttää, asettaa ja seurata. Lisäksi rajapinta määrittelee sille ominaisen virherajapinnan `DeviceStatusError`, jota kutsutaan silloin, kun ominaisuuden arvoa ei voida muuttaa. Virherajapinta perii `GenericError`-luokan. [17.]

GET-metodit

Metodi `getComponents` palauttaa kaikkien aspektiin liittyvien komponenttien nimet `StringArray`-tyypin tietona. Metodille annetaan parametri `AspectName`, jolla määritellään, minkä aspektin komponentit listataan. Mikäli sanastoa ei ole määritelty, käytetään oletussanastoa. Jos sanastolla tai aspektilla ei ole pätevää arvoa, metodi palauttaa `DeviceAPIError`-virheen koodilla `INVALID_ARGUMENT_ERROR`. Jos aspekti on pätevä, mutta laite ei ole vielä ottanut käyttöön sitä, metodi palauttaa `DeviceAPIError`-virheen koodilla `NOT_FOUND_ERROR`. [17.] Metodia käytetään koodiesimerkin 1 mukaisella tavalla.

```
var displays = bondi.devicestatus.getComponents({aspect:'Display'});
```

Koodiesimerkki 1: Näyttöaspektin komponenttien listaus.

Metodi `getPropertyValue` hakee laitteen tilaominaisuuden arvon kullekin ominaisuudelle tyypillisen tietotyypin mukaan. Metodille annetaan parametrit sanasto, aspekti, komponentti ja ominaisuus. Parametreista vain ominaisuus on pakollinen. Mikäli sanastoa ei ole määritelty, käytetään oletussanastoa. Jos aspektia ei ole määritelty, käytetään ensimmäistä aspektia, jolla on samanniminen tilaominaisuus kuin määritelty ominaisuus. Jos komponenttia ei ole määritelty, käytetään aspektin ensisijaista komponenttia. Jos ominaisuudella ei ole pätevää arvoa, metodi palauttaa `DeviceAPIError`-virheen koodilla `INVALID_ARGUMENT_ERROR`. Jos ominaisuutta ei ole vielä implementoitu, metodi palauttaa `DeviceAPIError`-virheen koodilla `NOT_FOUND_ERROR`. Mikäli turvallisuuskäytäntö epää pääsyn rajapintaan, metodi palauttaa `SecurityError`-virheen koodilla `PERMISSION_DENIED_ERROR`. [17.] Metodia käytetään koodiesimerkin 2 mukaisella tavalla.

```
var level = bondi.devicestatus.getPropertyValue({
     property:"currentOrientation",
     aspect:"Display",
     component:"__active"
});
```

Koodiesimerkki 2: Laitteen ensisijaisen näytön orientaation haku.

SET-metodit

Metodi `setDefaultVocabulary` asettaa oletussanaston. Metodille annetaan parametri sanasto, jonka täytyy näkyä `listVocabularies`-metodin palauttamassa sanastolistassa. Jos annettua sanastoa ei löydy, `setDefaultVocabulary`-metodi palauttaa `DeviceAPIError`-virheen koodilla `NOT_FOUND_ERROR`. BONDI-sanastoa tarjotaan oletusarvoisesti kyseiselle metodille. Mikäli sanastoa ei määritetä ollenkaan, metodi palauttaa `DeviceAPIError`-virheen koodilla `INVALID_ARGUMENT_ERROR`. [17.] Metodia käytetään koodiesimerkin 3 tavalla.

```
bondi.devicestatus.setDefaultVocabulary("http://bondi.omtp.org");
```

Koodiesimerkki 3: Oletussanastoksi asetetaan BONDI-sanasto.

Metodi `setPropertyValue` asettaa ominaisuudelle uuden arvon. Metodille annetaan parametrit sanasto, aspekti, komponentti, ominaisuus ja uusi arvo. Parametreista ominaisuus on pakollinen. Metodi `setPropertyValue` käyttäytyy samalla tavalla kuin `getPropertyValue` puuttuvien parametrien ja turvallisuuskäytännön suhteen. Edellä mainittujen lisäksi metodi palauttaa `DeviceStatusError`-virheen koodilla `READ_ONLY_PROPERTY_ERROR`, mikäli ominaisuus on vain luettavissa muttei kirjoitettavissa. [17.] Metodia käytetään koodiesimerkin 4 tavalla.

```
1 bondi.devicestatus.setPropertyValue({property:"currentOrientation",
2 aspect:"Display"}, 0);
```

Koodiesimerkki 4: Näytön orientaationasetus asentoon 0, eli pystysuoraan.

Listausmetodit

Metodi `listVocabularies` palauttaa saatavilla olevat sanastot, `listAspects` saatavilla olevat aspektit ja `listProperties` saatavilla olevat ominaisuudet. Kaikki listat ovat `StringArray`-tyyppisiä.

```
bondi.devicestatus.listVocabularies();
```

Koodiesimerkki 5: Saatavilla olevien sanastojen listaus.

Tarkkailumetodit

Metodi `watchPropertyChange` tarkkailee ominaisuuden muutoksia ja lähetettää niistä ilmoituksia. Metodille annetaan parametrit sanasto, aspekti, komponentti, ominaisuus, callback-funktio ja valinnat. Parametreista ominaisuus ja callback-funktio ovat pakollisia. Aina kun tarkkaillun ominaisuuden arvo muuttuu, järjestelmä kutsuu callback-funktiota ja välittää sille eteenpäin ominaisuuden uuden arvon. Valintaparametrilla määritellään ilmoitusten herkkyys. Sallittuja parametreja ovat `minTimeout` ja `maxTimeout`, joilla määritellään minimi- ja maksimiajat, joiden jälkeen pyyntö katkaistaan, `callCallbackOnRegister`, jolla kutsutaan callback-funktiota heti, kun se on rekisteröity, ja `minChangePercent`, joka määrittelee minimimuutoksen, jonka

ylittyessä ilmoitus lähetetään. Parametri `minChangePercent` pätee vain, jos tarkkailtavan ominaisuuden tietotyyppi on numeerinen. [17.]

Metodi palauttaa `watchHandler`-tyypin käsittelijän. Mikäli turvallisuuskäytäntö epää pääsyn ominaisuuksiin, metodi palauttaa `SecurityError`-virheen koodilla `PERMISSION_DENIED_ERROR`. Jos ominaisuudella ei ole pätevää arvoa, metodi palauttaa `DeviceAPIError`-virheen koodilla `INVALID_ARGUMENT_ERROR`, tai jos ominaisuutta ei ole vielä otettu käyttöön, metodi palauttaa `DeviceAPIError`-virheen koodilla `NOT_FOUND_ERROR`. [17.]

Koodiesimerkissä 6 asetetaan metodi tarkkailemaan näytön orientaation muutoksia. Muutoksien tapahtuessa kutsutaan alert-funktiota kaksi kertaa. Ensimmäisellä kerralla funktio näyttää käyttäjälle haetun ominaisuuden, komponentin, aspektin ja sanaston. Toisella kerralla näytetään uusi arvo. Lopuksi määritellään katkaisun aikarajaksi 1 000 millisekuntia ja kutsutaan callback-funktiota rekisteröitäessä.

```
1  var orientChangeHandler = bondi.devicestatus.watchPropertyChange(
2      {property:"currentOrientation", aspect:"Display"},
3      {
4          onPropertyChange:function(ref, value){
5              alert("Property changed: " + ref.property +
6                    " " + ref.component + " " + ref.aspect +
7                    " " + ref.vocabulary);
8              alert("New value: " + value);
9          }
10     },
11     {minTimeout:1000, callCallbackOnRegister:true}
12 );
```

Koodiesimerkki 6: Näytön orientaation tarkkailu.

Metodin `clearPropertyChange` avulla voidaan lopettaa ominaisuuden tarkkailu. Metodille annetaan parametri `watchHandler`, jolla määritellään, mikä käsittelijä poistetaan. Jos käsittelijää ei ole olemassa, metodi palauttaa `DeviceAPIError`-virheen koodilla `INVALID_ARGUMENT_ERROR`. [17.] Metodia käytetään koodiesimerkin 7 tavalla.

```
bondi.devicestatus.clearPropertyChange(orientChangeHandler);
```

Koodiesimerkki 7: Näytön orientaation tarkkailun lopetus.

Paikannusmoduuli

BONDIn paikannusmoduuli perustuu W3C:n paikannusrajapintoihin. Paikannusmoduuli mahdollistaa mobiililaitteen paikantamisen usealla eri paikantamismenetelmällä, kuten satelliittipaikannuksen, langattoman lähiverkon, puhelinverkon tai IP-osoitteen avulla. Myös näiden menetelmien yhdistelmiä voidaan käyttää. Paikannus voi olla joko kertaluontoista tai jatkuvaa. Paikannusmoduulin käyttö sallitaan pienoisohjelman alustustiedoston `feature`-elementissä nimellä `http://bondi.omtp.org/api/geolocation`. Paikannusmoduuli koostuu callback-funktioista (`PositionSuccessCallback`, `PositionErrorCallback`) paikannusrajapinnasta (`geolocation`), sijaintirajapinnasta (`position`), koordinaattirajapinnasta (`coordinates`), sijaintivirherajapinnasta (`PositionError`) ja sijaintivalinnat-rajapinnasta (`PositionOptions`). [18.]

Onnistuneen sijaintipyynnön callback-funktiota `PositionSuccessCallback` kutsutaan, kun kertaluontoinen tai jatkuva paikannus onnistuu. Callback-funktio taas kutsuu `handleEvent`-metodia, jolle annetaan parametri sijaintirajapinnan määrittelemässä muodossa. Epäonnistuneen sijaintipyynnon callback-funktiota `PositionErrorCallback` kutsutaan, kun kertaluontoinen tai jatkuva paikannus epäonnistuu. Callback-funktio kutsuu `handleEvent`-metodia, jolle annetaan sijaintivirherajapinnan mukainen parametri. [18.]

Paikannusrajapinta (`geolocation`) tarjoaa kolme metodia sijainnin hakemiseen. Metodi `getCurrentPosition` palauttaa asynkronisesti kertaluontoisen sijaintidatan. Metodille annetaan parametrit `successCallback`, `errorCallback` ja `options`. Parametri `successCallback` on `PositionSuccessCallback`-funktio, jota kutsutaan, kun sijainti on päivitetty onnistuneesti. Parametri `errorCallback` on `PositionErrorCallback`-funktio, jota kutsutaan, kun sijainnin päivitys epäonnistuu. Parametri `options` on

sijaintivalinnat-rajapinnan mukainen objekti, joka määrittelee sijaintipyyntöön liittyviä asetuksia. [18.] Metodia käytetään koodiesimerkin 8 mukaisesti.

```
bondi.geolocation.getCurrentPosition(onPositionSuccess,
onPositionError, options);
```

Koodiesimerkki 8: Sijainnin hakeminen.

Seurantametodi `watchPosition` hakee käyttäjän sijainnin ja päivittää sitä jatkuvasti. Seurantametodille annetaan samat parametrit kuin `getCurrentPosition`-metodille, mutta se palauttaa tunnisteen, jonka avulla seuranta voidaan poistaa. [18.] Seuranta-metodia käytetään koodiesimerkin 9 esittämällä tavalla.

```
bondi.geolocation.watchPosition(onPositionSuccess, onPositionError,
options);
```

Koodiesimerkki 9: Sijainnin seuranta.

Metodi `clearWatch` poistaa sijainnin seurannan. Metodille annetaan parametrina `watchPosition`-metodin palauttama tunniste, jolla määritellään juuri kyseisen seurannan poistaminen. [18.] Metodia käytetään koodiesimerkin 10 mukaisesti.

```
bondi.geolocation.clearWatch(id);
```

Koodiesimerkki 10: Seurannan poistaminen.

Sijaintirajapinta (`position`) määrittelee kaikki tiedot sijainnista ja asettaa sijaintidatalle aikaleiman. Sijaintirajapinnalle on määritelty attribuutit `timestamp` ja `coords`. Attribuutti `timestamp` on aikaleima, joka kertoo, kuinka monta sekuntia on kulunut tammikuun 1. päivästä vuonna 1970 sijaintipyynnön ajankohtaan. Aikaleima voidaan tarvittaessa muuttaa useisiin helpommin luettaviin päivämäärää ja aikaa kuvaaviin muotoihin. Attribuutti `coords` on koordinaattirajapinnan mukainen esitys sijaintidatasta. [18.]

Koordinaattirajapinta (`coordinates`) määrittelee attribuutit `latitude`, `longitude`, `altitude`, `accuracy`, `altitude accuracy`, `heading` ja `speed`. Attribuutti `latitude`

on sijainti leveysasteiden mukaan ja `longitude` pituusasteiden mukaan. Attribuutti `altitude` on sijainnin korkeus merenpinnasta. Attribuutti `accuracy` määrittelee, kuinka monen metrin tarkkuudella leveys- ja pituusasteiden sijainti haetaan. Attribuutti `altitude accuracy` taas määrittelee, kuinka monen metrin tarkkuudella sijainnin korkeus haetaan. Attribuutti `heading` on muuttuvan sijainnin suunta asteina ja `speed` muuttuvan sijainnin nopeus metreinä per sekunti. [18.]

Sijaintivirherajapinta (`PositionError`) määrittelee attribuutin `code`, joka antaa virhetapahtuman virhekoodin, ja attribuutin `message`, joka antaa virhetapahtuman kuvauksen. Sijaintivalinnat-rajapinta (`PositionOptions`) määrittelee valintoja, joilla voidaan vaikuttaa sijaintipyyntöihin. Attribuutti `timeout` määrittelee, minkä ajan päästä sijainnin pyyntö katkaistaan, `maximumAge` määrittelee, kuinka kauan edellinen sijaintidata on voimassa ja `enableHighAccuracy` asettaa sijaintipyynnölle paremman tarkkuuden. [18.]

Muut BONDI-ohjelmointirajapinnat

BONDI-moduuli (`bondi`) sisältää määritelmiä, jotka ovat yhteisiä kaikille moduuleille. Sovelluksen avausmoduuli (`bondi.applauncher`) määrittelee pääsyn asennettuihin sovelluksiin ja mobiililaitteen natiiveihin sovelluksiin, kuten puhelutekniikkaan. Lisäksi se tarjoaa toiminnallisuuden mainittujen sovelluksien käynnistämiseen. Sanomanvälitysmoduuli (`bondi.messaging`) määrittelee pääsyn laitteen viestijärjestelmään. Moduulin rajapintojen avulla voidaan lukea ja lähettää teksti-, multimedia- ja sähköpostiviestejä. Vuorovaikutusmoduuli (`bondi.ui`) määrittelee menetelmät, joilla sovellus voi olla vuorovaikutuksessa käyttäjän kanssa. Esimerkiksi mobiililaitteen värinää tai näytön orientaatiota voi hallita vuorovaikutusmoduulin rajapintojen avulla. [19; 20; 21; 22.]

Tiedostojärjestelmämoduuli (`bondi.filesystem`) määrittelee pääsyn mobiililaitteen tiedostojärjestelmään ja menetelmiä, joilla tiedostoja ja kansioita voidaan hallinnoida. Galleriamoduuli (`bondi.gallery`) määrittelee pääsyn mobiililaitteen mediakokoelmiin. Mediakokoelma voi sisältää kuva-, video- ja audiotiedostoja. Galleriamoduulin

rajapintojen avulla voidaan luoda mukautettuja näkymiä mediakokoelmista. Sovelluksen alustusmoduuli (`bondi.appconfig`) määrittelee menetelmiä, joilla haetaan tai muutetaan sovelluksen asetuksia, kuten sähköpostipalvelimen osoitetta. Kameramoduuli (`bondi.camera`) mahdollistaa kuvien ja videon ottamisen mobiililaitteen kameroilla. Moduuli tarjoaa myös menetelmiä, joilla kameroiden asetuksia voi muuttaa. [23; 24; 25; 26.]

Viestinnän lokimoduuli (`bondi.commlog`) määrittelee pääsyn viimeisimpien puhelujen tietoihin, kuten vastattuihin, soitettuihin ja vastaamatta jääneisiin puheluihin, sekä viimeisimpien tekstiviestien tietoihin, kuten lähetettyihin, vastaanotettuihin, luonnoksiin ja lähteviin viesteihin. Yhteystietomoduuli (`bondi.pim.contact`) määrittelee pääsyn mobiililaitteen tai SIM-kortin yhteystietoihin ja menetelmät yhteystietojen lisäämiseen, poistamiseen ja muokkaamiseen. Kalenterimoduuli (`bondi.pim.calendar`) määrittelee pääsyn mobiililaitteen kalentereihin ja tapahtumiin. Lisäksi moduuli tarjoaa menetelmiä tapahtumien lisäämiseen, poistamiseen ja muokkaamiseen. Tehtävämoduuli (`bondi.pim.task`) määrittelee pääsyn mobiililaitteeseen tallennettuihin tehtäviin. [27; 28; 29; 30.]

3.3 BONDI-pienoisohjelma

Pakkaus

BONDI-pienoisohjelmat ovat *zip*-arkistoja, jotka sisältävät kaikki pienoisohjelmaan liittyvät tiedostot. *Zip*-arkiston voi tehdä millä tahansa pakkausohjelmalla, joka tukee *zip*-tiedostomuotoa. BONDI-pienoisohjelmien rakenne ja sisältö perustuvat W3C:n pakkaus- ja alustusspesifikaatioon. BONDI-pienoisohjelman tiedostopäätteen tulee olla *wgt*, ja jos pienoisohjelmaa jaetaan Internetin välityksellä, se tulee merkitä `application/widget` Internet-mediatyypillä [31].

BONDI-pienoisohjelma voi sisältää niin monta tiedostoa, kuin on tarpeen. Taulukko 2 listaa varatut tiedostonimet, joilla on erityinen tehtävä pienoisohjelman toiminnallisuuden kannalta. Alustustiedosto sisältää tarvittavat asetukset ja tiedot

pienoisohjelman toimimiseen. Pienoisohjelmalle voi nimetä oman ikonin, jonka täytyy olla tyyppiä *png, gif, ico* tai *svg*. Jos omaa ikonia ei ole määritelty, käytetään oletuksena tiedostonimeä *icon.**, mikäli sellainen löytyy pienoisohjelmasta. Ikonit ovat vapaaehtoisia. Pienoisohjelmassa täytyy olla yksi aloitustiedosto, jonka on oltava tyyppiä *html, htm, svg, xhtml,* tai *xht*. Se voi olla joko itse nimetty tai sen puuttuessa oletusaloitustiedosto nimeltä *index.**. Näiden lisäksi pienoisohjelmassa voi olla kansio nimeltä *locales,* joka sisältää monikielisten pienoisohjelmien kielikohtaiset tiedostot.

Taulukko 2: BONDI-pienoisohjelman varatut tiedostonimet [31].

Tiedostonimi	Tarkoitus
config.xml	alustustiedosto
icon.png	oletusikoni
icon.gif	oletusikoni
icon.ico	oletusikoni
icon.svg	oletusikoni
index.html	oletusaloitustiedosto
index.htm	oletusaloitustiedosto
index.svg	oletusaloitustiedosto
index.xhtml	oletusaloitustiedosto
index.xht	oletusaloitustiedosto
locales	kansio lokalisoidulle sisällölle

Alustustiedosto

Alustustiedosto on XML-dokumentti, joka sisältää tarvittavat elementit pienoisohjelman asentamiseen ja alustamiseen. BONDI-pienoisohjelma saa sisältää vain yhden alustustiedoston, joka tulee nimetä muotoon *config.xml* ja jonka tulee sijaita arkiston juuressa. Alustustiedostossa on vain yksi pakollinen elementti `widget` ja attribuutti `xmlns`, jolloin tiedosto voi suppeimmillaan koostua koodiesimerkin 11 mukaisesta rivistä.

```
<widget xmlns="http://www.w3.org/ns/widgets" />
```

Koodiesimerkki 11: BONDIn suppein alustustiedosto [31].

Yleensä alustustiedosto koostuu kuitenkin useammasta elementistä. Koodiesimerkki 12 on tyypillinen BONDIn alustustiedosto. Koodiesimerkin riveillä 2–7 määritellään pienoisohjelman id, korkeus, leveys, versio ja näkymä. Rivillä 8 määritellään pienoisohjelman lyhyt ja pitkä nimi. Rivit 9–11 sallivat pienoisohjelman käyttää kameramoduulin ohjelmointirajapintoja ja asettavat kameran automaattisen tarkennuksen päälle. Riveillä 12–14 on määritelty nimi-arvopari, joka liitetään pienoisohjelmaan sen käynnistyessä ensimmäisen kerran. Rivit 15–18 kuvailevat pienoisohjelman toimintaa. Rivit 19–22 määrittelevät tietoja pienoisohjelman tekijästä. Rivi 23 määrittelee pienoisohjelman ikonin sijainnin ja nimen. Rivi 24 määrittelee aloitustiedoston nimen. Rivit 25–28 määrittelevät, millä lisenssillä pienoisohjelma on julkaistu.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <widget xmlns="http://www.w3.org/ns/widgets"
3          id="http://example.org/exampleWidget"
4          version="2.0 Beta"
5          height="200"
6          width="200"
7          viewmodes="widget">
8     <name short="Example 2.0">The example Widget!</name>
9     <feature name="http://bondi.omtp.org/api/camera">
10       <param name="autofocus" value="true"/>
11    </feature>
12    <preference name="apikey"
13                value="123456789a"
14                readonly="true" />
15    <description>
16       A sample widget to demonstrate
17       some of the possibilities.
18    </description>
19    <author href="http://foo-bar.example.org/"
20            email="foo-bar@example.org">
21       Foo Bar
22    </author>
23    <icon src="icons/example.png"/>
24    <content src="myWidget.html"/>
25    <license href="http://example.com/license">
26       Software is provided "as is"
27       without warranty of any kind.
28    </license>
29 </widget>
```

Koodiesimerkki 12: BONDIn alustustiedosto [31].

3.4 Kehitystyökalut

BONDI-pienoisohjelmien kehityksen aloituskynnys ei ole kovin korkea. XHTML-, CSS- ja JavaScript-taidot ovat siinä eduksi. Pienoisohjelmia voidaan kehittää ilman erikoistyökaluja tai kehitysympäristöjä, esimerkiksi pelkällä tekstieditorilla pärjää. Tekstieditoreista kannattaa kuitenkin valita sellainen, joka näyttää rivinumerot ja syntaksinkorostuksen. Useimmissa tietokoneen käyttöjärjestelmissä on myös arkistointisovellus, joka pystyy pakkaamaan tiedostoja *zip*-muotoon. Mikäli käyttöjärjestelmän mukana ei tule arkistointisovellusta, erikseen asennettavista arkistointisovelluksista avoimen lähdekoodin *7-zip* on tarkoitukseen mainiosti soveltuva ohjelmisto.

Kehityksen tehostamiseksi BONDI SDK -laajennus on saatavilla Eclipse-kehitysalustaan. Eclipsen BONDI SDK -laajennus ei kuitenkaan tue kaikkia BONDI-ohjelmointirajapintoja. BONDI SDK on myös saatavilla Windows Mobile 6.1 -käyttöjärjestelmän sisältäviin mobiililaitteisiin. Mobiililaitteille suunnattu BONDI SDK tukee useampia ohjelmointirajapintoja, kuitenkin laitteen ominaisuuksien puitteissa. JavaScript-vianetsintään erikoistunut *Mozilla Firefox* -selaimeen asennettava laajennus nimeltä *Firebug* on myös oiva lisä pienoisohjelmien kehitykseen ja testaukseen.

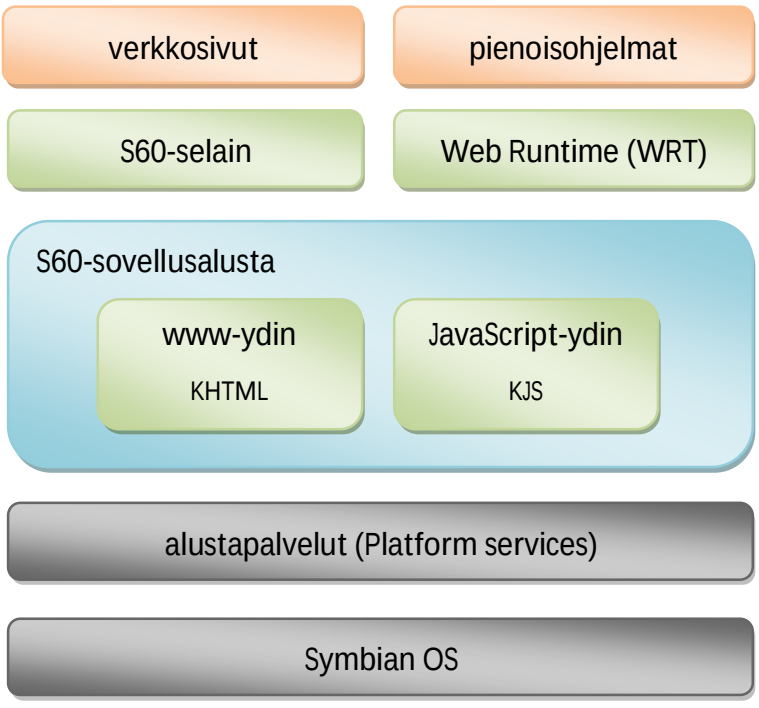
4 Nokian WRT-alusta

4.1 WRT-arkkitehtuuri

Web Runtime (WRT) on Nokian kehittämä pienoisohjelma-alusta, joka sisältyy S60-sovellusalustaan. WRT mahdollistaa pienoisohjelmien ja www-sovellusten kehityksen mobiiliympäristöön perinteisten www-teknologioiden avulla. WRT noudattaa seuraavia www-teknologioiden standardeja: HTML 4.01, XHTML MP, CSS 2.1 ja JavaScript 1.5 [32]. Kehittäjän ei siis tarvitse osata Symbian-ohjelmoinnissa käytettyä C++-kieltä luodakseen sovelluksia Symbian OS -käyttöjärjestelmällä varustettuihin mobiililaitteisiin.

Symbian OS tarjoaa sovelluksille pääsyn laitteen toimintoihin ja ominaisuuksiin alustapalveluiden (Platform Services) kautta. WRT tukee kahta alustapalveluiden rajapintatoteutusta: Platform Services 1.0 ja 2.0. WRT:n rajapinnat on toteutettu JavaScriptilla. Alustapalveluita voivat käyttää myös muut sovellukset, kuten Flash Lite, jonka rajapinnat on toteutettu ActionScriptilla. Tässä insinöörityössä käytettiin toteutusta Platform Services 1.0 ja WRT:n versiota 1.1. [33.]

Kuvassa 10 on esitetty WRT-ympäristön teknologiarakenne. Alimpana on Symbian OS, joka on Symbian Ltd:n kehittämä käyttöjärjestelmä älypuhelimia ja kämmentietokoneita varten. Symbian OS tarjoaa rajapintoja, joilla sovellukset voivat käyttää laitteen toimintoja ja ominaisuuksia. Rajapintoja kutsutaan alustapalveluiksi (Platform Services).



Kuva 10: WRT-arkkitehtuuri [34].

Nokian kehittämä S60 on käyttöliittymä ja sovellusalusta, joka on rakennettu Symbian OS -käyttöjärjestelmän päälle. S60 sisältää www-ytimen, joka vastaa HTML-, XML-, DOM-, ECMAScript-, JavaScript- ja CSS-kielien käsittelystä ja esittämisestä. Www-

ydin perustuu KHTML-ytimeen. S60 sisältää www-ytimen lisäksi myös JavaScript-ytimen, joka on erikoistunut JavaScriptin käsittelyyn erityisesti selaimissa. JavaScript-ydin perustuu KJS-ytimeen.

S60-sovellusalustan päälle on rakennettu S60-selain, joka on verkkosivujen selaukseen tarkoitettu ohjelmisto, ja WRT, joka sisältää tarvittavat komponentit pienoisohjelmien käyttöä ja hallintaa varten. WRT-arkkitehtuurin lopputuotteita ovat verkkosivut, joita selataan S60-selaimen kautta, ja pienoisohjelmat, joita suoritetaan WRT-ympäristössä.

4.2 WRT-ohjelmointirajapinnat

Yleistä WRT-ohjelmointirajapinnoista

WRT:n Platform Services 1.0 jaetaan kymmeneen toiminnallisuuksien mukaan ryhmiteltyyn palveluun. Palvelut ovat joukkoja toisiinsa liittyviä rajapintoja. Platform Services 1.0 -rajapintojen tarjoamia palveluja käytetään palveluobjektin (service object) avulla. Palveluobjekti luodaan `device.getServiceObject`-metodin avulla. Metodille annetaan parametrit `provider`, joka määrittelee, minkätyyppinen palveluobjekti luodaan, ja `interface`, joka määrittelee, mitä palveluntarjoajan rajapintaa käytetään. Metodi palauttaa onnistuessaan palveluobjektin ja epäonnistuessaan `undefined`. Koodiesimerkissä 13 on esitetty tapa, jolla palveluobjekti luodaan. Attribuuttien `provider` ja `interface` tilalle laitetaan jokaiselle palvelulle ominainen nimi ja rajapinta. [33.]

```
var so = device.getServiceObject(provider, interface);
```

Koodiesimerkki 13: Palveluobjektin luominen.

Tässä insinöörityöraportissa käsitellään tarkemmin järjestelmätietopalvelua (`System Info`) ja paikannuspalvelua (`Location`), koska ne ovat oleellisia työn kannalta. Muut palvelut esitellään yleisellä tasolla.

Järjestelmätieto

Järjestelmätieto antaa pienoisohjelmille mahdollisuuden hakea ja muokata järjestelmän tietoja. Lisäksi sen avulla on myös mahdollista seurata muuttuvia arvoja ja ilmoittaa muutoksista käyttäjälle. Järjestelmätietoja voidaan käyttää palveluobjektin avulla, joka luodaan koodiesimerkin 14 mukaisella tavalla. [35.]

```
var so = device.getServiceObject("Service.SysInfo", "ISysInfo");
```

Koodiesimerkki 14: Järjestelmätiedon palveluobjekti.

Järjestelmätietopalvelu tarjoaa neljä metodia, joilla tietoja voi käyttää. Metodi `ISysInfo.GetInfo` hakee järjestelmäattribuutin tietoja joko synkronisesti tai asynkronisesti. Synkronisesti käytettäessä metodille annetaan parametri `criteria`, joka määrittelee, mistä järjestelmäattribuutista haetaan tietoa. Asynkronisesti käytettäessä metodille annetaan `criterian` lisäksi parametri `callback`, jota kutsutaan, kun metodi palauttaa järjestelmäattribuutin arvon tai muutoksen. `Callback`-funktio tulee luoda itse. `ISysInfo.GetInfo` palauttaa synkronisessa kutsussa objektin, joka sisältää haetun järjestelmäattribuutin tiedot, virhekoodin ja virhekuvauksen. Asynkronisessa kutsussa metodi palauttaa objektin, joka sisältää tapahtuman tunnisteen, virhekoodin ja virhekuvauksen. Varsinaiset järjestelmäattribuutin tiedot palauttaa callback-funktio. [36.] Metodia käytetään koodiesimerkin 15 mukaisesti.

```
var result = so.ISysInfo.GetInfo(criteria);
```

Koodiesimerkki 15: Tiedon hakeminen synkronisesti.

Metodi `ISysInfo.SetInfo` asettaa järjestelmäattribuutille uuden arvon asynkronisesti. On huomioitava, että kaikille järjestelmäattribuuteille ei voi asettaa arvoa itse. Metodille annetaan parametri `criteria`, joka määrittelee järjestelmäattribuutin uuden arvon. Metodi palauttaa virhekoodin ja virhekuvauksen. [37.] Uusi arvo asetetaan koodiesimerkin 16 mukaisesti.

```
var result = so.ISysInfo.SetInfo(criteria);
```

Koodiesimerkki 16: Tiedon muuttaminen.

Metodi `ISysInfo.GetNotification` on asynkroninen metodi, joka ilmoittaa käyttäjälle, kun järjestelmäattribuutin arvo muuttuu. Metodille annetaan parametrit `criteria`, joka määrittelee, mitä järjestelmäattribuuttia seurataan, ja `callback`, jota kutsutaan, kun arvo muuttuu. Metodi palauttaa objektin, joka sisältää tapahtuman tunnisteen, virhekoodin ja virhekuvauksen. `Callback`-funktio palauttaa varsinaisen uuden arvon. Huomioitavaa on, että metodi seuraa muutoksia koko ajan, kunnes se peruutetaan metodilla `ISysInfo.Cancel`, ja ettei kaikkia järjestelmäattribuutteja voi seurata. [38.] Seuranta luodaan koodiesimerkin 17 mukaisesti.

```
var result = so.ISysInfo.GetNotification(criteria, callback);
```

Koodiesimerkki 17: Seurannan luonti.

`ISysInfo.Cancel` on synkroninen metodi, joka peruuttaa yllä mainitun metodin luoman seurannan. Metodille annetaan parametri `criteria`, joka sisältää tapahtuman tunnisteen. Tunnisteen avulla voidaan peruuttaa tietty seuranta. Metodi palauttaa objektin, joka sisältää virhekoodin ja virhekuvauksen. [39.] Seuranta peruutetaan koodiesimerkin 18 esittämällä tavalla.

```
var result = so.ISysInfo.Cancel(criteria);
```

Koodiesimerkki 18: Seurannan peruuttaminen.

Kaikissa metodeissa käytettävä `criteria`-objekti määrittelee, minkä järjestelmäattribuutin tietoja haetaan. Objektilla on kolme ominaisuutta: `Entity`, `Key` ja `SystemData`. `Entity` on pakollinen ominaisuus, joka määrittelee komponentin, josta tietoa haetaan. `Key` on pakollinen ominaisuus, joka määrittelee, mitä tietoa haetaan. Yhdessä `Entity` ja `Key` muodostavat järjestelmäattribuutin. `SystemData`-objekti sisältää joukon ominaisuuksia, jotka määrittelevät järjestelmäattribuutin arvon. `SystemData` on joko vapaaehtoinen tai pakollinen parametri `GetInfo`-metodille riippuen haettavasta

järjestelmäattribuutista. Koodiesimerkissä 19 on esitetty yksi tapa määritellä `criteria`-objekti, jota käytetään näytön orientaatioon liittyvissä kutsuissa. [40.]

```
1  var criteria = new Object();
2  criteria.Entity = "Display";
3  criteria.Key = "DisplayOrientation";
```

Koodiesimerkki 19: Criteria-objekti.

Paikannus

Paikannuspalvelun avulla pienoisohjelmat voivat hakea laitteen maantieteellisen sijainnin ja suorittaa sijaintiin perustuvia laskelmia. Toisin kuin BONDIn paikannusmoduuli, joka voi määritellä sijainnin useasta eri järjestelmästä tai niiden yhdistelmästä, WRT:n paikannuspalvelu käyttää laitteen oletuspaikannusmenetelmää. Paikannusdataa käytetään palveluobjektin avulla, joka määritellään koodiesimerkin 20 mukaisella tavalla. [41.]

```
var so = device.getServiceObject("Service.Location", "ILocation");
```

Koodiesimerkki 20: Paikannuksen palveluobjekti.

Paikannuspalvelu tarjoaa neljä metodia, joilla paikannusdataa voi käyttää. Metodi `ILocation.GetLocation` hakee kertaluontoisesti laitteen sijainnin joko synkronisesti tai asynkronisesti. Metodille annetaan parametri `criteria`, joka määrittelee, minkätyyppistä sijaintidataa haetaan ja miten. Asynkronisessa kutsussa metodille annetaan `criterian` lisäksi parametri `callback`, jota kutsutaan, kun sijainti muuttuu. `Callback`-funktio tulee luoda itse. Metodi palauttaa synkronisesti käytettynä objektin, joka sisältää sijaintidatan, virhekoodin ja virhekuvauksen. Asynkronisesti kutsuttuna metodi palauttaa objektin, joka sisältää tapahtuman tunnisteen, virhekoodin ja virhekuvauksen. Varsinaisen sijaintidatan palauttaa `callback`-funktio. Sijaintidatan saatavuus riippuu laitteen GPS-teknologiasta. [42.] Metodia käytetään asynkronisesti koodiesimerkin 21 esittämällä tavalla.

```
var result = so.ILocation.GetLocation(criteria, callback);
```

Koodiesimerkki 21: Sijainnin hakeminen asynkronisesti.

Metodi `ILocation.Trace` hake sijainnin jatkuvasti ennalta määrätyn ajan välein. Metodi on asynkroninen, joten sille annetaan parametreiksi `criteria` ja `callback`. Metodi palauttaa objektin, joka sisältää tapahtuman tunnisteen, virhekoodin ja virhekuvauksen. `Callback`-funktio palauttaa varsinaisen sijaintidatan. Metodi hakee päivityksiä, kunnes se peruutetaan `ILocation.CancelNotification`-metodin avulla. Tämän takia voi olla vain yksi aktiivinen seuranta kerrallaan. [43.] Metodia käytetään koodiesimerkin 22 mukaisesti.

```
var result = so.ILocation.Trace(criteria, callback);
```

Koodiesimerkki 22: Jatkuva sijainnin päivitys.

Metodi `ILocation.Calculate` suorittaa synkronisesti matemaattisia laskelmia lähde- ja kohdesijainnin suhteen. Metodille annetaan parametri `criteria`, joka sisältää suoritettavan laskutoimituksen ja arvot, joita käytetään laskutoimituksessa. Laskennan `criteria`-objekti eroaa huomattavasti muista paikannusmetodien `criteria`-objekteista. [44.] Taulukossa 3 on lueteltu kaikki laskennan `criteria`-objektin ominaisuudet. Hakasuluissa olevat ominaisuudet ovat vapaaehtoisia.

Taulukko 3: Laskennan criteria-objektin ominaisuudet [45].

Ominaisuus	Kuvaus	Tyyppi	Arvo
`criteria.MathRequest`	suoritettava laskutoimitus	merkki-jono	"FindDistance" "FindBearingTo" "MoveCoordinates"
`criteria.DistanceParamSource`	lLähdesijainnin koordinaatit	objekti	objekti, joka sisältää alla olevat arvot
`criteria.DistanceParamSource. Longitude`	lähdesijainnin pituusaste	numero	[+180.00, –180.00]
`criteria.DistanceParamSource. Latitude`	lähdesijainnin leveysaste	numero	[+90.00, –90.00]
`criteria.DistanceParamSource. Altitude`	lähdesijainnin korkeus	numero	kokonais- tai desimaaliluku
`[criteria.DistanceParamDestination]`	kohdesijainnin koordinaatit	objekti	objekti, joka sisältää alla olevat arvot
`[criteria.DistanceParamDestination. Longitude]`	kohdesijainnin pituusaste	numero	[+180.00, –180.00]
`[criteria.DistanceParamDestination. Latitude]`	kohdesijainnin leveysaste	numero	[+90.00, –90.00]
`[criteria.DistanceParamDestination. Altitude]`	kohdesijainnin korkeus	numero	kokonais- tai desimaaliluku
`[criteria.MoveByThisDistance]`	lähde- ja kohdesijainnin välimatka metreinä	numero	kokonais- tai desimaaliluku
`[criteria.MoveByThisBearing]`	lähdesijainnin muutoksen suunta asteina	numero	kokonais- tai desimaaliluku

Metodi `ILocation.Calculate` palauttaa objektin, joka sisältää laskutoimituksen tuloksen, virhekoodin ja virhekuvauksen [44]. Laskentametodia käytetään koodiesimerkin 23 mukaisesti.

```
var result = so.ILocation.Calculate(criteria);
```

Koodiesimerkki 23: Sijaintidatan laskutoimituksen luonti.

`ILocation.CancelNotification` on synkroninen metodi, joka peruuttaa aktiivisen sijainnin seurannan. Metodille annetaan parametri `criteria`, joka sisältää vain `CancelRequestType`-ominaisuuden. Ominaisuuden arvo voi olla joko `GetLocCancel`, joka peruuttaa kertaluontoisen sijainnin haun, tai `TraceCancel`, joka peruuttaa jatkuvan sijainnin seurannan. Metodi palauttaa objektin, joka sisältää virhekoodin ja virhekuvauksen. [46.] Metodia käytetään koodiesimerkin 24 mukaisesti.

```
var result = so.ILocation.CancelNotification(criteria);
```

Koodiesimerkki 24: Sijainnin jatkuvan päivityksen peruuttaminen.

Metodeissa `ILocation.GetLocation` ja `ILocation.Trace` käytetyllä `criteria`-objektilla on kaksi pääominaisuutta: `LocationInformationClass` ja `Updateoptions`. `LocationInformationClass`-ominaisuudella on kaksi vaihtoehtoa. Ensimmäinen on `BasicLocationInformation`, joka hakee sijainnin pituus-, leveys- ja korkeusdatan. Toinen on `GenericLocationInfo`, joka hakee kaiken mahdollisen sijaintidatan laitteen ominaisuuksien puitteissa. Sijaintidataan kuuluu pituus-, leveys- ja korkeusdatan lisäksi muun muassa nopeus, suunta ja satelliittien määrä. [47.]

`Updateoptions` määrittelee päivitysvaihtoehdot sijaintia haettaessa. Vaihtoehdot ovat `UpdateInterval`, `UpdateTimeOut`, `UpdateMaxAge` ja `PartialUpdates`. `UpdateInterval` määrittelee aikavälin kahden sijaintipäivityksen välillä mikrosekunteina. `UpdateTimeOut` määrittelee mikrosekunteina ajan, jonka sisällä sijaintidata tulisi hakea. Mikäli sijaintidataa ei haeta määritellyn ajan sisällä, metodi palauttaa `SErrTimedOut`-virheen. `UpdateMaxAge` määrittelee mikrosekunneissa, kuinka kauan sijaintidata on voimassa. `PartialUpdates` määrittelee, onko perussijaintidata (pituus, leveys ja korkeus) aina taattu. Vaihtoehto `false` takaa, että käyttäjä saa vähintään perussijaintidatan. Vaihtoehto `true` ei takaa päivityksiä mistään datasta.

`PartialUpdates`-ominaisuudesta on hyötyä, jos saatavilla on vain muutamia sijaintitietoja. [47.]

Jos päivitysvaihtoehtoja ei määritellä, käytetään oletusarvoja. Vaihtoehdot tulee määritellä suuruusjärjestyksessä, eli aikakatkaisun aikamäärän tulee olla suurempi kuin päivitysten aikaväli ja päivitysten aikavälin tulee olla suurempi kuin päivityksen voimassaoloaika. Jos järjestystä ei noudateta, metodi palauttaa `SErrArgument`-virheen. [47.]

Muut WRT-ohjelmointirajapinnat

Sovelluksen hallinta (`AppManager`) mahdollistaa sovelluksien käyttämisen ja käynnistämisen pienoisohjelman kautta. Sen avulla voi myös listata käyttäjän asentamat sovellukset tai kaikki (myös esiasennetut) sovellukset, käynnistää sovelluksen sen tunnisteen tai dokumentin Internet-mediatyypin perusteella. Kalenterin (`Calendar`) avulla käyttäjä voi luoda, käyttää ja hallita kalentereita ja kalenterimerkintöjä pienoisohjelman kautta. Yhteystietojen (`Contacts`) avulla voi käyttää ja hallita yhteystietoja pienoisohjelman kautta. Yhteystiedot voivat sijaita laitteen muistissa tai SIM-kortilla. [48; 49; 50.]

Maamerkkien (`Landmarks`) avulla voi käyttää ja hallita maamerkkejä ja niiden luokkia. Maamerkit ovat laitteen muistiin tallennettuja sijainti-nimipareja, joiden sijainti voidaan ilmaista koordinaatteina tai katuosoitteena. Maamerkit jaetaan luokkiin tyypin ja mielenkiinnonkohteiden mukaan. Loki (`Logging`) tarjoaa pääsyn laitteen lokitietoihin, kuten puhelujen tai viestien lokeihin. Niitä voi lisätä, poistaa ja muokata. Median hallinnan (`Media Management`) avulla pienoisohjelma voi hakea ja toistaa laitteella sijaitsevia mediatiedostoja, kuten kuvia, ääntä ja videota. [51; 52; 53.]

Viestintä (`Messaging`) tarjoaa mahdollisuuden lähettää, hakea ja hallita teksti- ja multimediaviestejä. Antureiden (`Sensors`) avulla pienoisohjelma voi hakea dataa laitteessa olevista antureista. Anturin data yhdistetään yhteen tai useampaan

anturikanavaan, joita rajapinta voi seurata. Anturikanavat ovat fyysisten antureiden abstraktioita. [54; 55.]

4.3 WRT-pienoisohjelma

Pakkaus

WRT-pienoisohjelmat ovat *zip*-arkistoja, jotka sisältävät kaikki pienoisohjelmaan liittyvät tiedostot. WRT-pienoisohjelmien pakkausrakenne eroaa BONDI-pienoisohjelmista, koska pienoisohjelma-arkiston tulee sisältää kansio, jossa kaikki pienoisohjelmaan liittyvät tiedostot sijaitsevat. Tiedostot eivät siis saa sijaita arkiston juuressa. WRT-pienoisohjelman tiedostopäätteen tulee olla *wgz*, ja jos pienoisohjelmaa jaetaan Internetin välityksellä, se tulee merkitä *x-nokia-widgets* Internet-mediatyypillä. [12, s. 37.]

Taulukossa 4 on lueteltu kaikki WRT-pienoisohjelmassa käytettävät tiedostot. Ensimmäisenä on alustustiedosto *info.plist*, joka sisältää pienoisohjelman asennukseen ja alustukseen tarvittavat asetukset. Alustustiedosto on pakollinen, ja sen on sijaittava kansion juuressa. Myös oletusaloitustiedosto on pakollinen, ja sen on sijaittava kansion juuressa. Oletusaloitustiedosto voi olla nimetty itse, mutta tiedoston tulee olla tyyppiä *html*. Tyyli-, skripti- ja kuvatiedostot ovat valinnaisia ja voivat sijaita joko kansion juuressa tai alikansioissa.

Taulukko 4: WRT-pienoisohjelman sisältämät tiedostot [56].

Tiedosto	Käyttö	Sijainti
info.plist	pakollinen	kansion juuressa
*.html	pakollinen	kansion juuressa
icon.png	valinnainen	kansion juuressa
*.css	valinnainen	kansion juuressa tai alikansiossa
*.js	valinnainen	kansion juuressa tai alikansiossa
*.jpg/bmp/gif/png	valinnainen	kansion juuressa tai alikansiossa

Alustustiedosto

WRT-pienoisohjelman alustustiedosto on XML-dokumentti, jonka tulee noudattaa Nokian määrittelemää DTD:tä (Document Type Definition). DTD on XML-kielen yhteydessä käytetty rakennemäärittelytapa, joka varmistaa, että XML:n rakenne on yhtenäinen kaikissa DTD:tä käyttävissä dokumenteissa. WRT-pienoisohjelman alustustiedosto tulee nimetä muotoon *info.plist*. Pienoisohjelmassa voi olla vain yksi alustustiedosto, ja sen tulee sijaita *zip*-arkistossa olevan kansion juuressa. [57.]

Koodiesimerkki 25 on WRT-pienoisohjelman alustustiedosto, joka sisältää kaikki mahdolliset elementit. WRT-pienoisohjelman *info.plist* on huomattavasti suppeampi kuin BONDIn alustustiedosto. Rivillä 1 on asetettu XML:n versio ja merkistö. Riveillä 2–3 on määritelty, että XML-koodin tulee olla Nokian DTD:n mukainen. Rivillä 4 on XML-dokumentin juurielementti `plist`, joka sisältää kaikki muut elementit. Juurielementissä määritellään alustustiedoston versio. Alustusdata on määritelty nimi-arvopareilla, joista `key` vastaa nimeä ja `string` arvoa. Nimielementtien sisällön tulee olla aina samassa muodossa kuin koodiesimerkissä 25. Pakollisia nimi-arvopareja ovat `DisplayName`, `Identifier` ja `MainHTML`. Nimi-arvoparien järjestys on vapaa.

Rivit 6–7 määrittelevät pienoisohjelman nimen. Rivit 8–9 määrittelevät pienoisohjelman yksilöllisen tunnisteen, joka on yleensä tekijän verkko-osoite käänteisessä muodossa. Rivit 10–11 määrittelevät oletusaloitustiedoston nimen. Rivit 12–13 määrittelevät pienoisohjelman version. Rivit 14–15 sallivat tai estävät pienoisohjelman verkkoyhteyden. Rivit 16–17 mahdollistavat pienoisohjelman näyttämisen aloitusnäytössä. Mikäli mobiililaite ei tue aloitusnäytön pienoisohjelmia, parametri ohitetaan ja pienoisohjelma toimii tavanomaisesti koko näytön tilassa.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!DOCTYPE plist PUBLIC "-//Nokia//DTD PLIST 1.0//EN"
3     "http://www.nokia.com/NOKIA_COM_1/DTDs/plist-1.0.dtd">
4  <plist version="1.0">
5     <dict>
6        <key>DisplayName</key>
7        <string>The example Widget!</string>
8        <key>Identifier</key>
9        <string>org.example.exampleWidget</string>
10       <key>MainHTML</key>
11       <string>Main.html</string>
12       <key>Version</key>
13       <string>1.0</string>
14       <key>AllowNetworkAccess</key>
15       <false/>
16       <key>MiniViewEnabled</key>
17       <true/>
18    </dict>
19 </plist>
```

Koodiesimerkki 25: WRT-pienoisohjelman alustustiedosto [57].

4.4 Kehitystyökalut

Kuten BONDI-pienoisohjelmien tapauksessa, WRT-pienoisohjelmien kehityksen voi myös aloittaa käyttämällä yksinkertaista tekstieditoria ja arkistointisovellusta. Molemmat pienoisohjelmat käyttävät samoja www-teknologioita, minkä takia niitä voi tehdä samoilla työkaluilla. Forum Nokia tarjoaa kuitenkin kehitystä helpottavia laajennuksia seuraaviin kehitysohjelmistoihin: Aptana Studio, Adobe Dreamweaver ja Microsoft Visual Studio [58, s. 7]. Laajennukset mahdollistavat pienoisohjelmien luomisen, testaamisen, pakkaamisen ja käyttöönoton.

Erityisesti testauskäyttöön Forum Nokia tarjoaa emulaattoreita, jotka simuloivat oikean matkapuhelimen käyttäytymistä. Emulaattoreita on tarjolla seuraavista alustoista: S60 3rd Ed. FP2, joka vastaa kolmannen sukupolven älypuhelimia, S60 5th Ed., joka vastaa kosketusnäytöllisiä älypuhelimia ja Nokia N97 SDK, joka simuloi Nokian N97-mallin matkapuhelinta. Lisäksi Forum Nokia ylläpitää Remote Device Access -palvelua, jonka avulla voi käyttää oikeita matkapuhelimia Internetin välityksellä. [58, s. 7.]

5 BONDI-pienoisohjelman rajapintojen kääntäminen Nokian WRT-alustalle

5.1 Tavoitteet ja rajaukset

Forum Nokia tarjosi mahdollisuuden tutkia ja toteuttaa insinöörityönä menetelmä, jolla BONDI-pienoisohjelma saataisiin toimimaan Nokian WRT-ympäristössä. Projektin tuli selvittää erilaisia keinoja kääntää rajapintoja ja toteuttaa niistä otollisin esimerkkisovellusten avulla. Lisäksi projektiin kuului dokumentaation, työnkulun ja toimintamallien laatiminen. Nopean aikataulun takia päädyttiin kääntämään BONDIn lukuisista rajapinnoista yksi paikannusmoduulin rajapinta ja yksi laitteen tilamoduulin rajapinta.

Työhön kuului kolme sovellusesimerkkiä pienoisohjelman kääntämisestä. Pienoisohjelmiksi valittiin BONDIn pienoisohjelmagalleriasta löytyvät esimerkkisovellukset, joista toinen ei käyttänyt BONDI-rajapintoja ollenkaan ja toinen käytti paikannusmoduulin rajapintoja. Pienoisohjelmagalleriasta ei löytynyt sopivaa pienoisohjelmaa, joka käyttäisi laitteen tilamoduulin rajapintoja, joten sellainen pienoisohjelma kehitettiin ensin BONDI-ympäristöön, minkä jälkeen se käännettiin WRT-ympäristöön.

Lähtökohtana oli, ettei pienoisohjelman alkuperäistä koodia tarvitsisi muokata ollenkaan, tai sitä tarvitsisi muokata mahdollisimman vähän. Yksi vaihtoehto oli luoda Symbian OS -käyttöjärjestelmän alustapalveluiden BONDI-versio, joka tarjoaisi samat toiminnallisuudet BONDI-metodien ja -objektien nimillä. Alustapalveluiden kehitys

olisi kuitenkin vaatinut Symbian-ohjelmoinnin opettelua, jota insinöörityön aikataulu ei sallinut. Tämän takia päätettiin kehittää JavaScript-käärekirjasto, joka simuloi BONDI-metodeja WRT-ympäristössä. Käärekirjastoa käytettäessä tarvitsee vain lisätä HTML-tiedostoihin referenssi kyseiseen käärekirjastoon, eikä pienoisohjelman toiminnallisuuden kannalta oleelliseen JavaScript-koodiin tarvitse koskea.

5.2 Kehitysympäristö ja -laitteet

JavaScript-käärekirjaston ohjelmistokehityksessä ja testauksessa käytettiin aluksi Eclipse-ympäristöä ja siihen asennettavaa BONDI SDK -laajennusta, mutta laajennuksen huomattiin olevan puutteellinen paikannusrajapintojen ja laitteen tilarajapintojen osalta. Lopulta päädyttiin käyttämään Eclipseä ohjelmistokehitykseen ja matkapuhelimiin asennettavaa BONDI SDK:ta testaukseen. Käytössä oli kolme mobiililaitetta, joista kaksi oli BONDI-pienoisohjelmien testaukseen tarkoitettuja ja yksi Nokian WRT-pienoisohjelmien testausta varten.

Vuoden 2009 lopussa vain Windows Mobile -käyttöjärjestelmän sisältävät matkapuhelimet tukivat BONDI-pienoisohjelmia, joten työssä käytettiin BONDI-yhteensopivia matkapuhelinmalleja HTC TyTN II ja Samsung SGh-i900 Omnia. HTC-matkapuhelimessa oli Windows Mobile -käyttöjärjestelmä, joka oli päivitetty versiosta 6.0 versioon 6.1. Samsung-matkapuhelimessa oli Windows Mobile 6.1 valmiina. WRT-pienoisohjelmia varten käytettiin Nokian N97-matkapuhelinta, jossa on S60 5th Ed. -käyttöliittymä ja Symbian OS -käyttöjärjestelmän versio 9.4. Käyttöjärjestelmä sisältää WRT:n version 1.1.

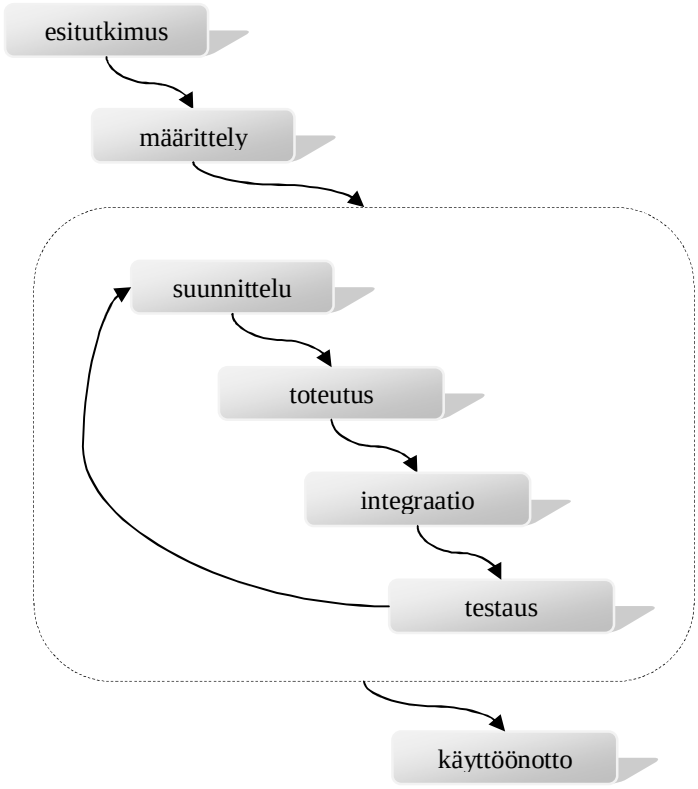
Pienoisohjelmien toiminnallisuutta testattiin myös Nokian Remote Device Access -palvelun avulla. Palvelu tarjoaa pääsyn useisiin matkapuhelinmalleihin Internetin välityksellä. Palvelun matkapuhelimet ovat fyysisiä laitteita eivätkä emulaattoreita.

5.3 Rajapintojen kääntämisprosessi

JavaScript-käärekirjaston kehityksessä sovellettiin elementtejä perinteisen ohjelmistokehityksen vesiputousmallista ja ketterän ohjelmistokehityksen menetelmistä. Kuvan 11 malli kuvaa lopullista menetelmää, jolla insinöörityötä vietiin eteenpäin. Mallin esitutkimusvaiheessa Forum Nokia selvitti, mitä ominaisuuksia JavaScript-käärekirjastolla tulisi olla. Lisäksi kehittäjä tutustui projektissa käytettäviin teknologioihin ja tekniikoihin. Määrittelyvaiheessa sovittiin Forum Nokian ja kehittäjän kesken ominaisuuksien tärkeysjärjestys, eli mitkä niistä otetaan mukaan työhön. Projektin aikataulutus ja tarvittava materiaali sisältyivät myös määrittelyvaiheeseen.

Mallissa katkoviiva kuvaa inkrementtiä, joka tässä projektissa tarkoittaa käännettävää rajapintamoduulia. Käännettävien rajapintamoduulien määrä oli rajoitettu kahteen. Inkrementti koostuu useista iteraatioista, joita toistetaan, kunnes inkrementti täyttää sille asetetut vaatimukset. Iteraation vaiheet ovat suunnittelu, toteutus, integraatio ja testaus. Suunnitteluvaiheessa vertailtiin BONDI- ja WRT-rajapintojen eroja ja suunniteltiin, kuinka rajapinnan kääntäminen olisi parasta toteuttaa. Seuraavaksi toteutettiin kääntäminen ja integroitiin syntynyt koodi olemassa olevan koodin kanssa. Integroinnin jälkeen testattiin käännetyn rajapinnan lisäksi myös aiemmin käännetyt rajapinnat varmistaen, ettei uusin koodi estä aiemman toimintaa.

Inkrementtien valmistumisen jälkeen viimeisenä vaiheena oli käyttöönotto. JavaScript-käärekirjasto julkaistiin Forum Nokian verkkosivuilla käyttöohjeiden, esimerkkisovellusten ja laajentamisohjeiden kera.



Kuva 11: Insinöörityössä käytetty suunnittelumalli.

5.4 Sovellusesimerkit

Pienoisohjelma, joka ei käytä BONDI-rajapintoja

Työssä käytettiin BONDIn pienoisohjelmagalleriasta löytyvän *Test widget* -nimisen pienoisohjelman versiota 1.1. Pienoisohjelma sisälsi alustustiedoston ja kaksi HTML-tiedostoa. Aloitustiedostossa oli kolme erilaista tapaa siirtyä toiseen HTML-tiedostoon: JavaScript-uudelleenohjaus, perinteinen HTML-linkki ja lomake. Toisessa tiedostossa testattiin `alert`-funktiota ja `history`-objektia. Objektin metodeilla `history.back` ja `history.go(-1)` siirryttiin yksi askel taaksepäin selaushistoriassa, eli takaisin aloitustiedostoon.

Pienoisohjelma asennettiin ensin Samsung SGh-i900 Omnia -matkapuhelimeen ja testattiin, että se toimii. Windows Mobile 6.1 ei kuitenkaan tukenut JavaScriptin `history`-metodeja, mutta itse pienoisohjelma toimi. Tämän jälkeen pienoisohjelman

tiedostopääte nimettiin *wgt*:stä takaisin *zip*-muotoon, jolloin pakkauksen voi avata arkistointiohjelmalla. Pakkaus purettiin, minkä jälkeen pienoisohjelman alustustiedosto *config.xml* avattiin. Tiedostosta poimittiin arvot kohdille id, versio ja nimi. Sen jälkeen luotiin WRT-pienoisohjelman alustustiedosto *info.plist* ja lisättiin sinne versio ja nimi. Id:n arvo oli URL-muotoinen, jota ei voi käyttää sellaisenaan WRT:n alustustiedostossa. Vakiintuneen tavan mukaan URL muutettiin käänteiseen muotoon ja lisättiin alustustiedoston `Identifier`-kohtaan.

Tässä vaiheessa pienoisohjelma sisälsi seuraavat tiedostot: *config.xml*, *info.plist* ja kaksi HTML-tiedostoa. BONDIn alustustiedostolla ei ollut enää merkitystä, joten sen olisi voinut halutessaan poistaa. Se kuitenkin jätettiin referenssiksi. Tiedostot siirrettiin kansioon, joka pakattiin *zip*-arkistoksi. Arkiston tiedostopääte nimettiin uudelleen *wgz*-muotoon. Pienoisohjelma siirrettiin ja asennettiin Nokian N97-matkapuhelimeen, minkä jälkeen sitä testattiin.

Pienoisohjelma, joka käyttää paikannusmoduulin rajapintaa

Työssä käytettiin BONDIn pienoisohjelmagalleriasta löytyvää Here I am! -pienoisohjelman versiota 1.0. Pienoisohjelma sisältää paikannuksen lisäksi myös muita toiminnallisuuksia, kuten sijainnin tallentamisen tiedostoon, sijainnin lukemisen tiedostosta ja sijainnin tallentamisen palvelimelle Ajax-tekniikan avulla. Työn kannalta oleellisia olivat kuitenkin vain paikannustoiminnot. Pienoisohjelma sisälsi seuraavat tiedostot: *config.xml*, *aplix.png*, *geo.html*, *geo.js*, *log.js* ja *style.css*.

Aloitustiedosto *geo.html* sisälsi kaksi tekstikenttää, joihin sijainnin pituus- ja leveyskoordinaatit tulostetaan, kun paikannus onnistuu. Kenttien alla oli viisi painiketta, joista kaksi liittyi paikannukseen. Ensimmäisestä painikkeesta haettiin sijainti ja toisesta näytettiin sijainti kartalla. Paikannuksen toiminnallisuus määriteltiin *geo.js*-tiedostossa. Tiedosto sisälsi funktiot `geohello`, jolla haettiin sijainti, `successCallback`, jota kutsutaan kun sijainnin haku onnistuu, `errorCallback`, jota kutsutaan, kun sijainnin haku epäonnistuu, ja `tile`, joka näyttää koordinaattien mukaan keskitetyn karttakuvan.

Pienoisohjelma asennettiin Samsung-matkapuhelimeen ja testattiin, että se toimii. Pienoisohjelma haki onnistuneesti sijainnin GPS-paikannuksen avulla ja näytti koordinaatit tekstikentissä. Myös kartta näytettiin keskitettynä koordinaattien sijainnin mukaan. Seuraavaksi luotiin toiminnallisuudeltaan vastaava WRT-pienoisohjelma, joka hakee sijainnin GPS:n avulla ja näyttää sen kartalla. Pienoisohjelman toiminnallisuus testattiin Nokian matkapuhelimessa. Kun molemmille ympäristöille oli olemassa toiminnallisuudeltaan samanlainen pienoisohjelma, vertailtiin niiden koodeja esimerkkisovellusten ja dokumentaatioiden avulla.

Koodiesimerkissä 26 ovat Here I am! -pienoisohjelman paikantamiseen käytetyt funktiot. Funktio `successCallback` (rivit 1–6) ottaa parametrin `p`, joka sisältää varsinaiset koordinaatit, ja asettaa koordinaatit tekstikenttien arvoiksi. Funktio `errorCallback` (rivit 8–12) näyttää virheilmoituksen `alert`-funktiolla. Funktio `geohello` (rivit 14–19) määrittelee sijainnin haun aikarajaksi 30 000 millisekuntia eli 30 sekuntia. Mikäli sijaintia ei ole saatu tämän ajan kuluessa, haku katkaistaan. Sijainnin haku suoritetaan riveillä 17–18.

```
1  function successCallback(p) {
2       lat = document.getElementById("lat");
3       lon = document.getElementById("lon");
4       lat.value = p.coords.latitude;
5       lon.value = p.coords.longitude;
6  }
7
8  function errorCallback(err) {
9       var msg = "Sorry, could not find a GPS signal! Do you have a
10 clear view of the sky?";
11      alert(msg);
12 }
13
14 function geohello() {
15      var options = {};
16      options.timeout = 30000;
17      bondi.geolocation.getCurrentPosition(successCallback,
18 errorCallback, options);
19 }
```

Koodiesimerkki 26: Here I am! -pienoisohjelman alkuperäinen koodi [59].

Koodiesimerkissä 27 on luotu BONDI-pienoisohjelmaa toiminnallisuudeltaan vastaavan WRT-pienoisohjelman paikannusfunktiot. Ensimmäisellä rivillä alustetaan palveluobjektin muuttuja. Funktio `callback` (rivit 3–15) eroaa BONDI-pienoisohjelman `callback`-funktioista, sillä se sisältää sekä onnistuneen että epäonnistuneen sijainnin haun toiminnallisuudet. Funktio ottaa parametrit `transId`, `eventCode` ja `result`. Parametri `result` sisältää virhekoodin ja sijainnin koordinaatit. Haun onnistuessa virhekoodi on `0`, jolloin rivillä 5 oleva `if`-ehtolause siirtyy suoraan `else`-kohtaan riville 9 ja täyttää tekstikentät koordinaattien arvoilla. Mikäli haussa tapahtuu virhe, on virhekoodi jotain muuta kuin `0`, jolloin ehtolauseen ensimmäinen osa suoritetaan, eli käyttäjälle näytetään virheilmoitus `alert`-funktiolla.

Funktio `geohello`, jonka avulla haetaan sijainti, alkaa riviltä 17. Rivillä 18 määritellään palveluobjekti nimenomaan paikannuksen palveluobjektiksi. Riveillä 19–21 luodaan `criteria`-objekti, joka määrittelee haun aikakatkaisun 30 000 000 mikrosekuntiin eli 30 sekuntiin. Rivillä 22 suoritetaan sijainnin haku.

```
1  var so;
2
3  function callback(transId, eventCode, result){
4       var errCode = result.ErrorCode;
5       if (errCode) {
6            var msg = "Sorry, could not find a GPS signal! Do you
7  have a clear view of the sky?";
8            alert(msg);
9       }else{
10           lat = document.getElementById("lat");
11           lon = document.getElementById("lon");
12           lat.value = result.ReturnValue.Latitude;
13           lon.value = result.ReturnValue.Longitude;
14      }
15 }
16
17 function geohello() {
18      so = device.getServiceObject("Service.Location", "ILocation");
19      var criteria = {};
20      criteria.Updateoptions = {};
21      criteria.Updateoptions.UpdateTimeOut = 30000000;
22      so.ILocation.GetLocation(criteria, callback);
23 }
```

Koodiesimerkki 27: BONDI-ympäristön paikannuspienoisohjelman toiminnallisuutta vastaava WRT-ympäristön koodi.

Ennen kuin kääntämistä oli aloitettu, asiakas ehdotti, että työssä käytettäisiin yhteisön kehittämää *platformservices.js*-nimistä JavaScript-kirjastoa. Kirjasto toimi kääreenä, joka muutti WRT:n paikannusrajapinnat W3C:n spesifikaation mukaisiksi. Kääre otettiin käyttöön, koska BONDIn paikannusrajapinta perustuu W3C:n rajapintoihin. Kääntämistä varten oli kuitenkin tarve luoda toinen kääre, koska BONDI-objekteja ja funktioita ei ole määritelty WRT-ympäristössä. Kääreen kehityksessä käytettiin JSON-muotoista objektien esitystapaa, koska sen katsottiin olevan selkolukuinen ja yksinkertainen. Koodiesimerkin 28 mukainen rakenne luotiin WRT-ympäristöön koodiesimerkin 29 esittämällä tavalla.

```
bondi.geolocation.getCurrentPosition(successCallback, errorCallback,
options);
```

Koodiesimerkki 28: BONDIn sijainnin hakufunktio.

```
1  var bondi = {
2       geolocation: {
3            getCurrentPosition:
4                 function(onPositionSuccess, onPositionError,
5  options) {
6                      //Toiminnallisuus tulisi tähän
7            }
8       }
9  };
```

Koodiesimerkki 29: Kääreen rakenne paikannusrajapinnan suhteen.

Toiminnallisuuden kehittämisessä alustettiin ensin palveluobjekti, joka sijaitsi `bondi`-objektin ulkopuolella, jotta muut palvelut voisivat käyttää samaa objektia. Koodiesimerkin 29 rivin 6 tilalle määriteltiin paikannuksen palveluobjekti, minkä jälkeen haettiin sijainti `platformservices`-kirjaston määrittelemällä paikannusfunktiolla. WRT:n paikannusfunktion parametreiksi annettiin samat parametrit, jotka annettiin BONDI-funktiota kutsuttaessa. Lopuksi toiminnallisuuskoodi ympäröitiin try-catch-virheenkäsittelijällä. Tulokseksi saatiin koodiesimerkin 30 mukainen koodi, joka tallennettiin *bondi.js*-nimiseksi JavaScript-käärekirjastoksi.

```
var serviceObj = null;

var bondi = {
   geolocation: {
      getCurrentPosition: function(onPositionSuccess,
                                   onPositionError,
                                   options)
      {
         try {
            if (!serviceObj) {
               serviceObj = com.nokia.device.load("",
               "com.nokia.device.geolocation");
            }

            serviceObj.getCurrentPosition(onPositionSuccess,
                                          onPositionError,
                                          options);
         }
         catch (ex) {
            alert("Error in geolocation: " + ex.message);
         }
      }
   }
};
```

Koodiesimerkki 30: Kääre, jossa on käännetty myös paikannustoiminnallisuus.

Alkuperäisen BONDI-pienoisohjelman HTML-tiedoston `head`-osioon lisättiin koodiesimerkin 31 mukaiset rivit, jotka mahdollistavat JavaScript-käärekirjastojen käytön pienoisohjelmassa. Muita muutoksia koodiin ei tarvinnut tehdä. JavaScript-käärekirjastot tallennettiin pienoisohjelman juureen. Pienoisohjelman juureen luotiin myös WRT:n alustustiedosto *info.plist*, johon poimittiin tiedot BONDIn alustustiedostosta. Pienoisohjelman ikonina toiminut *aplix.png* nimettiin uudelleen muotoon *icon.png*, koska WRT-ympäristö ei hyväksy muita muotoja.

```
<script type="text/javascript" src="bondi.js"></script>
<script type="text/javascript" src="platformservices.js"></script>
```

Koodiesimerkki 31: HTML-tiedostoon lisättävät rivit.

Tässä vaiheessa pienoisohjelma sisälsi seuraavat tiedostot: *config.xml*, *icon.png*, *info.plist*, *geo.html*, *platformservices.js*, *bondi.js*, *geo.js*, *log.js* ja *style.css*. Tiedostot lisättiin kansioon, joka pakattiin *zip*-arkistoksi. Arkiston tiedostopääte muutettiin

muotoon *wgz*, ja pienoisohjelma siirrettiin puhelimeen. Siirron jälkeen pienoisohjelma asennettiin ja testattiin.

Pienoisohjelma, joka käyttää laitteen tilamoduulin rajapintaa

Yhtä laitteen tilamoduulin rajapintaa käyttävää pienoisohjelmaa ei löytynyt BONDIn pienoisohjelmagalleriasta, joten sellainen luotiin tätä työtä varten. Laitteen tilamoduulin rajapinnoista valittiin akun lataustila. Yksinkertainen pienoisohjelma sisälsi tekstikentän ja ja painikkeen, josta tekstikentän sisältö muutettiin lataustilan mukaan. Mikäli laitteen akkua ladattiin, painiketta painamalla tekstikentän sisältö ilmaisi laitteen olevan latauksessa. Vastaavasti kun akkua ei ladattu, painiketta painamalla tekstikentän sisältö päivittyi ilmoittamaan, ettei laitetta ladata.

Pienoisohjelma asennettiin Samsung-matkapuhelimeen, ja sen toiminta testattiin. Testauksen jälkeen luotiin vastaavanlainen pienoisohjelma WRT-ympäristöön. Pienoisohjelmia verrattiin keskenään, minkä perusteella alettiin rakentaa *bondi.js*-kääreen toista osaa. Koodiesimerkissä 32 haetaan akun lataustila BONDI-ympäristössä. Ensin luodaan objekti `ref` (rivit 1–3), joka sisältää aspektin ja ominaisuuden, joilla määritellään, mitä tietoa haetaan. Lopuksi rivillä 5 haetaan tieto. Koodiesimerkissä 33 haetaan akun lataustila WRT-ympäristössä. Ensin luodaan laitteen tilamoduulin palveluobjekti (rivit 1–2). Sen jälkeen luodaan `criteria`-objekti (rivit 4–6), joka määrittelee, mitä tietoa haetaan. Rivillä 8 haetaan tieto. Ympäristöjen tavat hakea tilatietoja ovat periaatteeltaan samanlaisia.

```
1  var ref = {};
2  ref.aspect = "Battery";
3  ref.property = "batteryBeingCharged";
4
5  var value = bondi.devicestatus.getPropertyValue(ref);
```

Koodiesimerkki 32: BONDI-funktio, jolla haetaan akun lataustila.

```
1  var serviceObj = device.getServiceObject("Service.SysInfo",
2  "ISysInfo");
3
4  var criteria = new Object();
5  criteria.Entity = "Battery";
6  criteria.Key = "ChargingStatus";
7
8  var result = serviceObj.ISysInfo.GetInfo(criteria);
```

Koodiesimerkki 33: WRT-funktio, jolla haetaan akun lataustila .

Samaan tapaan kuin paikannusrajapinnan kääntämisessä, myös laitteen tilarajapinnan kääntämisessä käytettiin JSON-esitystapaa. Näin kääreen osat oli helpompi yhdistää myöhemmin. Koodiesimerkin 32 rivillä 5 on esitetty BONDI-funktio tiedon hakemiseen. Funktion rakennetta WRT-ympäristössä vastaa koodiesimerkissä 34 esitetty rakenne.

```
1  var bondi = {
2     devicestatus: {
3        getPropertyValue: function (ref) {
4           //Toiminnallisuus tulisi tähän
5        }
6     }
7  };
```

Koodiesimerkki 34: Kääreen rakenne tilarajapinnan suhteen.

Toiminnallisuutta lisättäessä ensin tuli alustaa palveluobjekti `bondi`-objektin ulkopuolelle, minkä jälkeen voitiin luoda siitä laitteen tilamoduulin palveluobjekti. Palveluobjektin luonti ympäröitiin try-catch-virheenkäsittelijällä. Sen jälkeen tarkistettiin, minkä aspektin ja ominaisuuden `ref`-objekti sisältää ja vastasivatko ne suunniteltuja arvoja. Tarkoituksena oli tukea vain akun lataustilaa, eli aspektia `Battery` ja ominaisuutta `BatteryBeingCharged`. Aspektia ja ominaisuutta WRT-ympäristössä vastaavat arvot lisättiin `criteria`-objektin `Entity`- ja `Key`-kohtiin. Sen jälkeen kutsuttiin metodia, joka hakee tilatiedon. Myös tilatiedon hakumetodi ympäröitiin try-catch-virheenkäsittelijällä. Tulokseksi saatiin koodiesimerkissä 35 esitetty koodi, joka tallennettiin *bondi.js*-käärekirjastoksi.

```
var serviceObj = null;

var bondi = {
    devicestatus: {
        getPropertyValue: function (ref) {
            try {
                serviceObj =
                device.getServiceObject("Service.SysInfo",
                "ISysInfo");
            } catch (ex) {
                alert("Service object cannot be found.");
            }

            var criteria = new Object();
            if(ref.aspect == "Battery"){
                criteria.Entity = "Battery";
                if (ref.property == "batteryBeingCharged"){
                    criteria.Key = "ChargingStatus";
                    try {
                        var result =
                        serviceObj.ISysInfo.GetInfo(criteria);
                        return result.ReturnValue.Status;
                    } catch (ex) {
                        alert("Error getting charging status: " +
                        ex);
                    }
                }
            }
        }
    }
};
```

Koodiesimerkki 35: Käännetty laitteen tilarajapinta.

Alkuperäisen BONDI-pienoisohjelman HTML-tiedoston `head`-osioon lisättiin koodiesimerkin 36 mukainen rivi. Muita muutoksia koodiin ei tarvinnut tehdä. JavaScript-käärekirjasto tallennettiin pienoisohjelman juureen. Jotta pienoisohjelma toimisi WRT-ympäristössä, sen juureen luotiin myös WRT:n alustustiedosto *info.plist*, johon poimittiin tiedot BONDIn alustustiedostosta. Poimitut tiedot olivat nimi, tunniste, versio ja aloitustiedosto.

```
<script type="text/javascript" src="bondi.js"></script>
```

Koodiesimerkki 36: HTML-tiedostoon lisättävä rivi.

Pienoisohjelman ikoni oli jo valmiiksi nimetty muotoon *icon.png*, joten sitä ei tarvinnut muuttaa. Tiedostot siirrettiin kansioon, joka pakattiin *zip*-arkistoksi. Arkiston tiedostopääte muutettiin *wgz*:ksi, minkä jälkeen pienoisohjelma siirrettiin ja asennettiin puhelimeen testausta varten.

5.5 Alustustiedostojen kääntäjä

Alustustiedostojen kääntäjän ominaisuuksien määrittely

Insinöörityöhön kuului käärekirjaston kehittämisen lisäksi myös alustustiedostojen automatisoidun kääntäjän suunnittelu ja toteutus. Tarkoitus oli luoda työkalu, joka helpottaa BONDI-, W3C- WRT- ja Opera-pienoisohjelmien alustustiedostojen kääntämistä keskenään. Kääntäjä toteutettiin PHP:n ja sen SimpleXML-laajennuksen avulla.

Kääntäjästä jätettiin tietoisesti pois osa W3C:n ja Operan spesifikaation mukaisista elementeistä ja attribuuteista, koska niille ei ollut vastaavia arvoja muissa ympäristöissä tai ne olivat harvinaisia. Tarvittaessa käyttäjä voi lisätä puuttuvat osat alustustiedostoon itse. W3C:n puolelta puuttuvat attribuutit ovat `xml:lang`, jonka avulla määritellään elementin ja sen sisällä olevien elementtien ja attribuuttien kieli, ja `its:dir`, joka määrittelee tekstin lukusuunnan. Elementeistä puuttuu `preference`, johon voidaan tallentaa tietoa nimi-arvopareina. Tietoa ei voi muuttaa pienoisohjelman ollessa käynnissä.

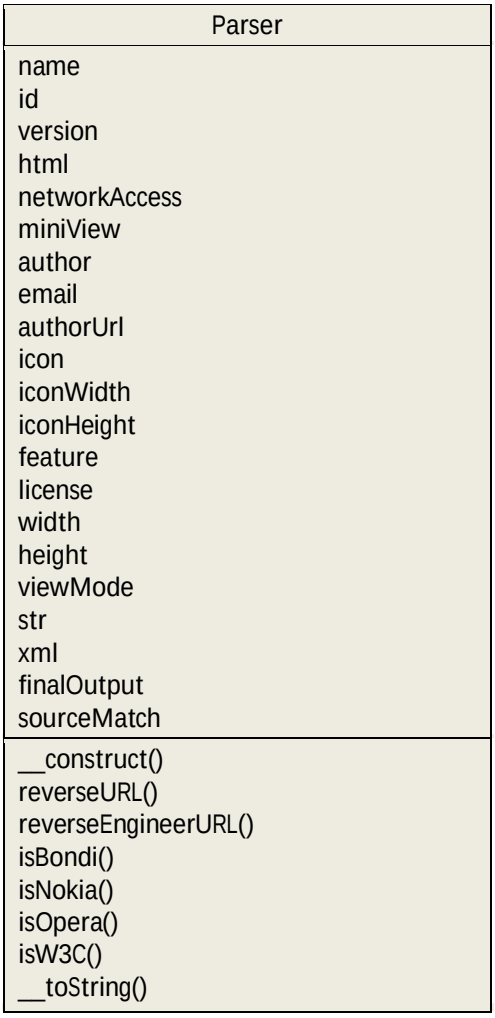
Operan puolelta puuttuvat elementit ovat `feature`, `param` ja `security`, joista kaksi ensimmäistä määrittelevät Opera-rajapintojen käytön ja niihin liittyvät valinnat. Viimeinen elementti `security` määrittelee sallitut protokollat, verkko-osoitteet ja portit. Operan turvallisuustoimenpiteet ovat huomattavasti yksityiskohtaisemmat kuin muiden ympäristöjen toimenpiteet, joten niillä ei olisi ollut vastaavia arvoja muualla.

Alustustiedostojen kääntäjän toteutus

Parser-luokka

Aluksi opiskeltiin BONDI-, Opera- ja WRT-ympäristöjen dokumentaatioita alustustiedoston osalta. Jokainen alustustiedosto oli erilainen, mutta BONDI ja Opera muistuttivat toisiaan eniten. Niiden kahden välillä kääntäminen onnistuisi suhteellisen helposti, koska lähes jokaisella BONDIn elementillä on vastaava elementti Operan alustustiedostossa. WRT:n alustustiedosto taas erosi muista sekä rakenteeltaan että toiminnallisuudeltaan. Se oli tiedoiltaan hyvin suppea ja rakenteeltaan nimi-arvomallinen. Nimi-arvomallissa ensimmäinen elementti sisältää toiminnallisuuden nimen ja sen alapuolella oleva elementti itse arvon. Muissa alustustiedostoissa rakenne on toteutettu niin, että elementti on nimetty suoraan toiminnallisuutta kuvaavalla nimellä ja sisältää myös arvon.

Eroavuuksien selvittelyn jälkeen luotiin kuvan 12 mukainen luokka `Parser`, joka sisältää muuttujia, joihin tieto väliaikaisesti tallennetaan, ja metodit `construct`, `isNokia`, `isW3C`, `isOpera`, `reverseURL`, `reverseEngineerURL` ja `toString`. Ensimmäinen metodi `construct` on rakentaja, jota kutsutaan uutta instanssia luotaessa. Rakentaja ottaa parametrin `data`, joka on lähdealustustiedoston XML-koodi, parametrin `source`, joka määrittelee, minkä ympäristön alustustiedostoa lähdetään kääntämään, ja parametrin `output`, joka määrittelee, minkä ympäristön alustustiedostoksi alkuperäinen tiedosto halutaan kääntää.



Kuva 12: Parser-luokan luokkakaavio.

Aluksi rakentaja siisti lomakkeesta lähetetyn XML-koodin poistamalla ylimääräiset vinoviivat, minkä jälkeen XML-koodista luotiin `SimpleXMLElement`-objekti. Objektin käsittely alkoi tarkistamalla `if`-ehtolauseiden avulla, vastaavatko XML-koodi ja valittu lähdemuoto toisiaan. Metodi `isNokia` palauttaa arvon `true`, mikäli lähdemuodoksi on valittu *info.plist* ja XML-koodi on WRT-määrittelyn mukainen. Metodit `isW3C` ja `isOpera` toimivat samalla periaatteella. Koska BONDIn alustustiedoston rakenne ja toiminnallisuus perustuvat W3C:n spesifikaatioon, voidaan tarkastaa sekä BONDIn että yleisten W3C-alustustiedostojen oikeellisuus `isW3C`-metodilla.

Jos kyseessä on *info.plist*, XML-koodi käydään läpi kahdesti ja siitä poimitaan tarvittavat arvot talteen muuttujiin. Sivulla 43 olevassa koodiesimerkissä 25 on esitetty

tyypillinen WRT-alustustiedosto. Kaikki nimi-arvoelementit on nimetty samalla tavalla, ja niiden järjestyskin voi olla satunnainen. Tämän takia XML-elementit tuli käydä läpi kahteen kertaan. Ensimmäisellä kerralla `key`-elementti merkittiin `boolean`-muuttujalla. Toisella kerralla tallennettiin merkityn elementin alla olevan elementin arvo. Kaksinkertainen läpikäyminen toistettiin jokaisen `dict`-elementin sisällä olevan elementin kohdalla. Näin nimi-arvoelementtiparien järjestyksellä ei ole merkitystä XML-koodin jäsentämisen kannalta. Jos kyseessä oli BONDIn (W3C:n) tai Operan alustustiedosto, tarvitsi XML-koodi käydä läpi vain kerran, koska siitä saatiin suoraan tallennettua arvot etsimällä XML-dokumentista tietyn elementin nimi ja sen sisältämä arvo.

Kun tiedot oli tallennettu muuttujiin, tarkistettiin `switch`-rakenteen avulla, mihin kohdemuotoon alustustiedosto tulisi kääntää. Mikäli lähde- ja kohdemuodot olivat samat, alkuperäistä XML-koodia ei käännetty vaan näytettiin sellaisenaan.

Jos kohdemuoto oli *info.plist*, kutsuttiin metodia `reverseURL` muuntamaan pienoisohjelman URL-muotoinen tunniste käänteiseen muotoon. Metodi ottaa verkkotunnuksen päätteen ensimmäiseksi merkkijonoksi, jonka perään se lisää verkkotunnuksen ja mahdollisen hakemistopolun pisteillä erotettuna. Esimerkiksi kyseinen metodi muuntaisi osoitteen `http://esimerkki.fi/widget` muotoon `fi.esimerkki.widget`. Verkkotunnuksen kääntämisen jälkeen muuttujiin tallennetut tiedot lisättiin `heredoc`-syntaksin mukaiseen *info.plist*-rakenteeseen, joka tallennettiin muuttujaan `finalOutput`.

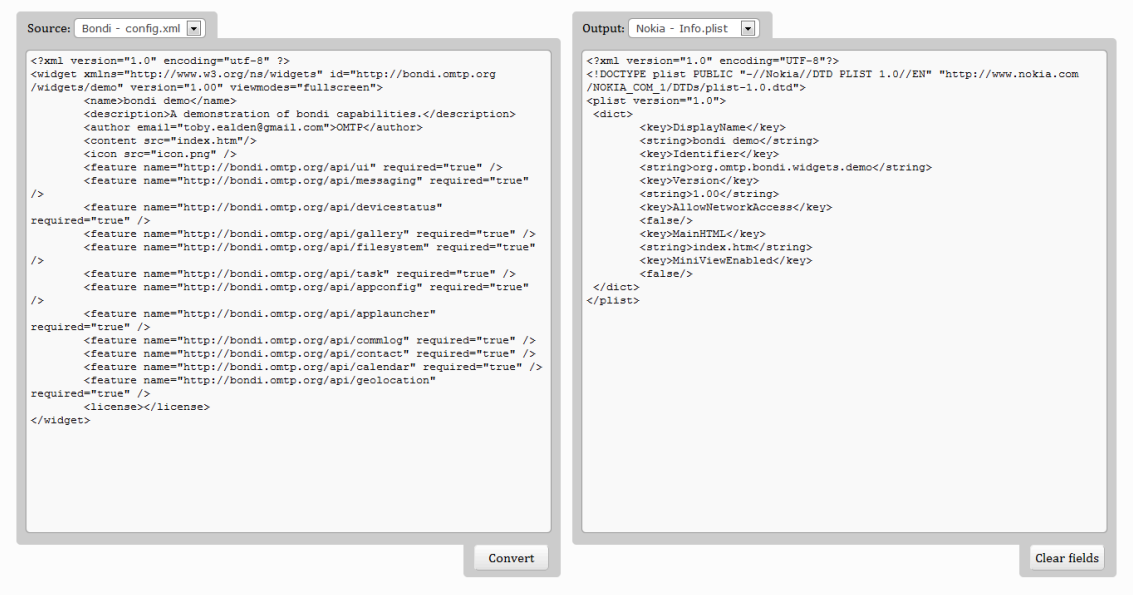
Jos kohdemuoto oli BONDIn tai W3C:n *config.xml*, tarkistettiin, oliko `viewmode`-attribuutilla lähde- ja kohdemuodoissa vastaavia arvoja. Mikäli ei ollut, käytettiin oletusarvoa `floating`. Sen jälkeen tarkistettiin pienoisohjelman `id`:n muoto. Jos se oli URL-muotoinen, arvolle ei tehty mitään. Jos `id` oli käänteisessä muodossa, siitä rakennettiin jälleen URL-muotoinen `reverseEngineerURL`-metodilla. Lopuksi muuttujiin tallennetut tiedot lisättiin `heredoc`-syntaksin mukaiseen BONDIn tai W3C:n alustustiedoston rakenteeseen, joka tallennettiin muuttujaan `finalOutput`. BONDI- ja W3C-alustustiedostot ovat samanlaiset `feature`-elementtejä lukuun ottamatta. BONDIn

`feature`-elementteihin on lisätty jokaisen BONDI-rajapinnan nimi, josta käyttäjä voi tarvittaessa poistaa käyttämättömät rajapinnat. W3C:n `feature`-elementti on tyhjä.

Kuten BONDIn tapauksessa, jos kohdemuoto oli Operan *config.xml*, tarkistettiin, oliko `defaultmode`-attribuutilla lähde- ja kohdemuodoissa vastaavia arvoja. Mikäli ei ollut, käytettiin oletusarvoa `widget`. Sen jälkeen tarkistettiin pienoisohjelman `id`-elementin arvo. Mikäli `id` oli käänteisessä muodossa, se muutettiin `reverseEngineerURL`-metodin avulla jälleen URL-muotoiseksi. Näiden lisäksi luotiin väliaikainen muuttuja, johon tallennettiin kuluva kuukausi ja vuosi, jotta ne voitiin lisätä `heredoc`-syntaksin mukaiseen Operan alustustiedoston rakenteeseen muiden tallennettujen tietojen ohella. Operan alustustiedostossa on `revised`-kohta, joka kertoo, koska pienoisohjelmaa on viimeksi päivitetty. Muutospäivämäärä luotiin automaattisesti, koska oletettiin pienoisohjelman valmistuvan kuluvan kuukauden sisällä. Alustustiedoston rakenne tallennettiin `finalOutput`-muuttujaan. Viimeinen metodi `toString` palauttaa `finalOutput`-muuttujassa olevan XML-koodin.

HTML-lomake

Kuvassa 13 on esitetty verkkosivu, jossa on yksi lomake. Lomake sisältää kaksi alasvetovalikkoa, joista toisesta valitaan alustustiedoston lähdemuoto ja toisesta halutun alustustiedoston kohdemuoto. Vaihtoehdot ovat samat molemmissa ”`Bondi - config.xml`”, ”`W3C - config.xml`”, ”`Opera - config.xml`” ja ”`Nokia - info.plist`”. Valikoiden alla on kaksi tekstikenttää, joista vasemmanpuoleiseen syötetään alkuperäisen pienoisohjelman alustustiedoston XML-koodi. Oikeanpuoleiseen tekstikenttään tulostuu käännetyn alustustiedoston XML-koodi. Tekstikenttien alapuolella on kaksi painiketta, joista toinen suorittaa kääntämisen ja toinen tyhjentää kenttien arvot.



Kuva 13: Alustustiedostojen kääntäjän käyttöliittymä.

Lomake on suunniteltu JavaScriptiä tukeville selaimille. JavaScriptin avulla tyhjennetään lomakkeen kentät ja säädetään lomakkeen korkeus vastaamaan selaimen korkeutta. Alustustiedostojen kääntäminen toimii myös ilman JavaScriptiä, mutta edellä mainitut käyttöliittymän ominaisuudet puuttuvat, jos JavaScript puuttuu. Lomakkeen ulkoasu on tehty CSS:n ja JavaScriptin avulla selaimen resoluutiosta riippumattomaksi. Lomake säilyttää alasvetovalikoiden arvot ja tulostetun XML-koodin, kunnes kentät tyhjennetään tai verkkosivulle navigoidaan uudelleen.

6 Yhteenveto

Insinöörityössä tutkittiin ja toteutettiin menetelmä, jolla BONDI-pienoisohjelmia voidaan käyttää Nokian WRT-ympäristössä. Menetelmän tueksi kehitettiin kolme esimerkkisovellusta, dokumentaatiot ja automaattinen alustustiedostojen kääntäjä. Menetelmäksi valittiin JavaScript-käärekirjasto, koska sillä voitiin simuloida BONDI-metodien ja -objektien rakennetta yksikertaisesti ja selkeästi. Käärekirjasto sisälsi kaksi käännettyä rajapintaa: paikannus- ja järjestelmätietorajapinnat. Projektin suunnitteluvaiheessa oli selvää, että JavaScript-käärekirjasto tultaisiin julkaisemaan avoimena lähdekoodina Forum Nokian verkkosivuilla, jotta muille kehittäjille annettaisiin mahdollisuus laajentaa kirjaston ominaisuuksia tulevaisuudessa. Kirjasto

julkaistiin BSD-lisenssillä, joka sallii koodin muokkaamisen ja uudelleenkäytön myös kaupallisessa tarkoituksessa.

Paikannusrajapintoja ja laitteen tilarajapintoja käyttävistä alkuperäisistä BONDI-pienoisohjelmista kehitettiin WRT-ympäristöön käännetyt esimerkkisovellukset käärekirjastojen avulla. Kolmanneksi esimerkkisovellukseksi valittiin pienoisohjelma, joka ei käyttänyt BONDI-rajapintoja. Käännetyt esimerkkisovellukset lisättiin Forum Nokian sivuille.

Dokumentaatiot sisälsivät vaiheittain etenevät ohjeet BONDI-pienoisohjelmien kääntämisestä. Myös käärekirjaston laajentamisesta kirjoitettiin ohjeet, jotka käyvät läpi BONDI- ja WRT-metodien eroja ja esittävät tässä insinöörityössä käytetyn käännöstavan. Lisäksi laajentamisohjeissa on myös erikseen lueteltu yleinen kehitysprosessi synkronisille ja asynkronisille metodeille, koska niitä käännettäessä on käytettävä hieman erilaisia lähtökohtia. Dokumentaatiot julkaistiin Forum Nokian verkkosivuilla käärekirjaston ja esimerkkisovellusten kera.

Käärekirjasto dokumentaatioineen sai huomattavasti näkyvyyttä verkossa ja oli hakukoneiden tuloksien kärjessä muun muassa hakusanoilla *widget porting* tai *porting bondi widgets*. Käärekirjasto toi lisää näkyvyyttä BONDI-hankkeelle ja edisti WRT:n ja BONDIn pienoisohjelmien yhteensopivuutta.

Alustustiedostojen kääntäjä yksinkertaisti BONDI-, W3C-, Opera ja WRT-alustustiedostojen kääntämisen. Sen avulla voitiin vain syöttää alkuperäinen XML-koodi ja saada siitä automaattisesti kohdeympäristön alustustiedosto alkuperäisen pienoisohjelman tiedoilla täytettynä.

Lähteet

1. What is BONDI? (WWW-dokumentti.) OMTP Limited. <http://bondi.omtp.org/whatisbondi/WHAT%20WE%20DO/Home.aspx>. 2010. Luettu 22.1.2010.

2. Cáceres, Marcos. Widgets 1.0: The Widget Landscape (Q1 2008). W3C Working Draft. (WWW-dokumentti.) W3C. <http://www.w3.org/TR/2008/WD-widgets-land-20080414/>. 14.4.2008. Luettu 18.1.2010.

3. Wells, Terri. Why Widgets Matter? (WWW-dokumentti.) <http://www.seochat.com/c/a/Search-Engine-News/Why-Widgets-Matter/>. 25.7.2007. Luettu 17.2.2010.

4. Swick, Ralph. & Ackerman, Mark. The X Toolkit: More Bricks for Building User-Interfaces, or, Widgets for Hire. USENIX. Massachusetts Institute of Technology, 1988.

5. Lal, Rajesh. Widgets for Web 2.0: What is a web widget? (WWW-dokumentti.) <http://www.widgets-gadgets.com/2007/08/what-is-web-widget.html>. 11.8.2007. Luettu 21.1.2010.

6. Cáceres, Marcos. Widgets 1.0 Requirements. W3C Working Draft. (WWW-dokumentti.) W3C. <http://www.w3.org/TR/2007/WD-widgets-reqs-20070705/>. 2007. Luettu 20.1.2010.

7. Mac 101: Dashboard. (WWW-dokumentti.) Apple Inc. <http://support.apple.com/kb/HT2492>. 2009. Luettu 22.1.2010.

8. Baudais, Eric et al. GUI Widgets. (WWW-dokumentti.) GNOME Documentation Project. <http://projects.gnome.org/gnumeric/doc/sect-graphics-widgets.shtml>. 2007. Luettu 1.2.2010.

9. What are Mobile Widgets? (WWW-dokumentti.) Feedzilla. <http://www.feedzilla.com/articles/widgets/what-are-mobile-widgets-/>. 18.9.2008. Luettu 1.2.2010.

10. Carson, Ryan. Meet Twiggy, our new mobile widget. (WWW-dokumentti.) Carsonified. <http://carsonified.com/blog/web-apps/meet-twiggy-our-new-mobile-widget/>. 18.3.2009. Luettu 2.2.2010.

11. Introduction to home screen widgets. (WWW-dokumentti.) Forum Nokia. <http://library.forum.nokia.com/index.jsp?topic=/Web_Developers_Library/GUID-DA3E6868-F769-4576-A0C3-6776BF358B44.html>. 2009. Luettu 2.2.2010.

12. Hakala, Jouni. Widget Development for S60 Web Run-Time. Master of Science Thesis. Tampere University of Technology, 2007.

13. OMTP - Open Mobile Terminal Platform. (WWW-dokumentti.) <http://www.omtp.org/About.aspx>. Luettu 22.1.2010.

14. BONDI Architecture and Security v1.0. (WWW-dokumentti.) OMTP Limited. <http://bondi.omtp.org/1.0/security/BONDI_Architecture_and_Security_v1.0.pdf>. 26.5.2009. Luettu 28.1.2010.

15. OMTP and BONDI. (WWW-dokumentti.) Betavine. <http://www.betavine.net/bvportal/partners/omtp>. Luettu 23.1.2010.

16. Coloma, Daniel & Hanclik, Marcin. Bondi API Specification - Version 1.0. (WWW-dokumentti.) OMTP. <http://bondi.omtp.org/1.0/apis/index.html>. 28.5.2009. Luettu 26.2.2010.

17. Arias, Mario et al. The Bondi Device Status Module: bondi.devicestatus - Version 1.0. (WWW-dokumentti.) OMTP. <http://bondi.omtp.org/1.0/apis/devicestatus.html>. 28.5.2009. Luettu 26.2.2010.

18. Paul, André & Krüssel, Steffen. The Bondi Geolocation Module: bondi.geolocation - Version 1.0. (WWW-dokumentti.) OMTP. <http://bondi.omtp.org/1.0/apis/geolocation.html>. 28.5.2009. Luettu 26.2.2010.

19. The bondi Module - Version 1.0. (WWW-dokumentti.) OMTP. <http://bondi.omtp.org/1.0/apis/bondi.html>. 28.5.2009. Luettu 26.2.2010.

20. Laliena, Guillermo & Tarancon, Pedro. The Bondi Application Launcher Module: bondi.applauncher - Version 1.0. (WWW-dokumentti.) OMTP. <http://bondi.omtp.org/1.0/apis/applauncher.html>. 28.5.2009. Luettu 26.2.2010.

21. Laliena, Guillermo et al. The Bondi Messaging Module: bondi.messaging - Version 1.0. (WWW-dokumentti.) OMTP. <http://bondi.omtp.org/1.0/apis/messaging.html>. 28.5.2009. Luettu 26.2.2010.

22. Ealden, Toby. The Bondi User Interaction Module: bondi.ui - Version 1.0. (WWW-dokumentti.) OMTP. <http://bondi.omtp.org/1.0/apis/ui.html>. 28.5.2009. Luettu 26.2.2010.

23. Byers, Paddy & Garbe, Anselm. The Bondi File System Module: bondi.filesystem - Version 1.0. (WWW-dokumentti.) OMTP. <http://bondi.omtp.org/1.0/apis/filesystem.html>. 28.5.2009. Luettu 26.2.2010.

24. Vercelli, Stefano. The Bondi Gallery Module: bondi.gallery - Version 1.0. (WWW-dokumentti.) OMTP. <http://bondi.omtp.org/1.0/apis/gallery.html>. 28.5.2009. Luettu 26.2.2010.

25. Ealden, Toby. The Bondi Application Configuration Module: bondi.appconfig - Version 1.0. (WWW-dokumentti.) OMTP. <http://bondi.omtp.org/1.0/apis/appconfig.html>. 28.5.2009. Luettu 26.2.2010.

26. Bassbouss, Louay et al. The Bondi Camera Module: bondi.camera - Version 1.0. (WWW-dokumentti.) OMTP. <http://bondi.omtp.org/1.0/apis/camera.html>. 28.5.2009. Luettu 26.2.2010.

27. Laliena, Guillermo et al. The Bondi Communication Log Module: bondi.commlog - Version 1.0. (WWW-dokumentti.) OMTP. <http://bondi.omtp.org/1.0/apis/commlog.html>. 28.5.2009. Luettu 26.2.2010.

28. Laliena, Guillermo & Tarancon, Pedro. The Bondi Contact Module: bondi.pim.contact - Version 1.0. (WWW-dokumentti.) OMTP. <http://bondi.omtp.org/1.0/apis/contact.html>. 28.5.2009. Luettu 26.2.2010.

29. Laliena, Guillermo & Tarancon, Pedro. The Bondi Calendar Module: bondi.pim.calendar - Version 1.0. (WWW-dokumentti.) OMTP. <http://bondi.omtp.org/1.0/apis/calendar.html>. 28.5.2009. Luettu 26.2.2010.

30. Laliena, Guillermo & Tarancon, Pedro. The Bondi Task Module: bondi.pim.task - Version 1.0. (WWW-dokumentti.) OMTP. <http://bondi.omtp.org/1.0/apis/task.html>. 28.5.2009. Luettu 26.2.2010.

31. Cáceres, Marcos. Widget Packaging and Configuration. (WWW-dokumentti.) W3C. <http://www.w3.org/TR/2009/CR-widgets-20091201/> 1.12.2009. Luettu 16.2.2010.

32. Web Runtime Standards. (WWW-dokumentti.) Forum Nokia. <http://library.forum.nokia.com/index.jsp?topic=/Web_Developers_Library/GUID-2786E26F-EF1E-4935-9C5F-56E36717E89D.html> 2009. Luettu 2.3.2010.

33. Using Platform Services 1.0 (WWW-dokumentti.) Forum Nokia. <http://library.forum.nokia.com/index.jsp?topic=/Web_Developers_Library/GUID-46EABDC1-37CB-412A-ACAD-1A1A9466BB68.html> 2009. Luettu 2.3.2010.

34. Salo, Kari. Yliopettaja, Metropolia Ammattikorkeakoulu, Espoo. Luentokalvot 24.3.2010.

35. Platform Services 1.0 System Information API (WRT 1.1). (WWW-dokumentti.) Forum Nokia. <http://library.forum.nokia.com/index.jsp?topic=/Web_Developers_Library/GUID-AEB26A58-1DE2-46CB-81EC-6DB3A477B7A3.html> 2009. Luettu 2.3.2010.

36. ISysInfo.GetInfo(). (WWW-dokumentti.) Forum Nokia. <http://library.forum.nokia.com/index.jsp?topic=/Web_Developers_Library/GUID-279A3698-D09B-44BF-8340-739E19F94727.html> 2009. Luettu 2.3.2010.

37. ISysInfo.SetInfo(). (WWW-dokumentti.) Forum Nokia. <http://library.forum.nokia.com/index.jsp?topic=/Web_Developers_Library/GUID-4347A7F1-1560-4B28-B4A8-DC570864F505.html> 2009. Luettu 2.3.2010.

38. ISysInfo.GetNotification(). (WWW-dokumentti.) Forum Nokia. <http://library.forum.nokia.com/index.jsp?topic=/Web_Developers_Library/GUID-7978FAFD-5BAC-45FD-9C44-5CF7BC484A1E.html> 2009. Luettu 2.3.2010.

39. ISysInfo.Cancel(). (WWW-dokumentti.) Forum Nokia. <http://library.forum.nokia.com/index.jsp?topic=/Web_Developers_Library/GUID-BF044B8D-C16A-47DC-84F3-9103D9DCE09B.html> 2009. Luettu 2.3.2010.

40. Criteria for retrieving system attribute information. (WWW-dokumentti.) Forum Nokia. <http://library.forum.nokia.com/index.jsp?topic=/Web_Developers_Library/GUID-D14E0D5C-2D99-41C8-8857-455409065500.html> 2009. Luettu 2.3.2010.

41. Platform Services 1.0 Location API. (WWW-dokumentti.) Forum Nokia. <http://library.forum.nokia.com/index.jsp?topic=/Web_Developers_Library/GUID-53CE4DE6-F065-4339-8C18-5C30A9540053.html> 2009. Luettu 2.3.2010.

42. ILocation.GetLocation(). (WWW-dokumentti.) Forum Nokia. <http://library.forum.nokia.com/index.jsp?topic=/Web_Developers_Library/GUID-150E440F-56E4-4249-8739-A5A7A2050600.html> 2009. Luettu 3.3.2010.

43. ILocation.Trace(). (WWW-dokumentti.) Forum Nokia. <http://library.forum.nokia.com/index.jsp?topic=/Web_Developers_Library/GUID-7A0190CB-545E-48D2-BAF0-D28E03CDFCA5.html> 2009. Luettu 3.3.2010.

44. ILocation.Calculate(). (WWW-dokumentti.) Forum Nokia. <http://library.forum.nokia.com/index.jsp?topic=/Web_Developers_Library/GUID-DBEE8177-7246-4FEE-A0F1-D6AEEA6EA206.html> 2009. Luettu 3.3.2010.

45. Calculation criteria. (WWW-dokumentti.) Forum Nokia. <http://library.forum.nokia.com/index.jsp?topic=/Web_Developers_Library/GUID-9115340A-5D15-4139-A236-945D199583AF.html> 2009. Luettu 3.3.2010.

46. ILocation.CancelNotification(). (WWW-dokumentti.) Forum Nokia. <http://library.forum.nokia.com/index.jsp?topic=/Web_Developers_Library/GUID-7A466B60-88DD-4A3A-B64C-E7300D42DA56.html> 2009. Luettu 3.3.2010.

47. Criteria for retrieving location information. (WWW-dokumentti.) Forum Nokia. <http://library.forum.nokia.com/index.jsp?topic=/Web_Developers_Library/GUID-4541E532-CC1A-4115-8467-7FA1C4378307.html> 2009. Luettu 3.3.2010.

48. Platform Services 1.0 AppManager API. (WWW-dokumentti.) Forum Nokia. <http://library.forum.nokia.com/index.jsp?topic=/Web_Developers_Library/GUID-2D651505-F68C-4053-B565-9FF826C5B897.html> 2009. Luettu 2.3.2010.

49. Platform Services 1.0 Calendar API. (WWW-dokumentti.) Forum Nokia. <http://library.forum.nokia.com/index.jsp?topic=/Web_Developers_Library/GUID-CABB28B8-D2B1-496B-BD7E-34FF496E60B4.html> 2009. Luettu 2.3.2010.

50. Platform Services 1.0 Contacts API. (WWW-dokumentti.) Forum Nokia. <http://library.forum.nokia.com/index.jsp?topic=/Web_Developers_Library/GUID-A5853293-7B83-4CCE-9C29-B2B8F0CD8A18.html> 2009. Luettu 2.3.2010.

51. Platform Services 1.0 Landmarks API. (WWW-dokumentti.) Forum Nokia. <http://library.forum.nokia.com/index.jsp?topic=/Web_Developers_Library/GUID-B8845FF1-D7F6-476A-8651-8B9C12D8789F.html> 2009. Luettu 2.3.2010.

52. Platform Services 1.0 Logging API. (WWW-dokumentti.) Forum Nokia. <http://library.forum.nokia.com/index.jsp?topic=/Web_Developers_Library/GUID-F003B7DD-E450-49AD-B447-C5132FE47D3C.html> 2009. Luettu 2.3.2010.

53. Platform Services 1.0 Media Management API. (WWW-dokumentti.) Forum Nokia. <http://library.forum.nokia.com/index.jsp?topic=/Web_Developers_Library/GUID-E5684FF0-28C9-4721-9421-6B9432087086.html> 2009. Luettu 2.3.2010.

54. Platform Services 1.0 Messaging API. (WWW-dokumentti.) Forum Nokia. <http://library.forum.nokia.com/index.jsp?topic=/Web_Developers_Library/GUID-91579EE5-A608-401C-82B4-DCF1723EC7B5.html> 2009. Luettu 2.3.2010.

55. Platform Services 1.0 Sensors API. (WWW-dokumentti.) Forum Nokia. <http://library.forum.nokia.com/index.jsp?topic=/Web_Developers_Library/GUID-B77C2006-879F-4AC6-B7BF-04B25B563A29.html> 2009. Luettu 2.3.2010.

56. Widget component files. (WWW-dokumentti.) Forum Nokia. <http://library.forum.nokia.com/index.jsp?topic=/Web_Developers_Library/GUID-0E3095DB-03FF-4240-83F2-6D876AD2083A.html> 2009. Luettu 3.3.2010.

57. Creating the info.plist file. (WWW-dokumentti.) Forum Nokia. <http://library.forum.nokia.com/index.jsp?topic=/Web_Developers_Library/GU

ID-BBA0299B-81B6-4508-8D5B-5627206CBF7B.html> 2009. Luettu 3.3.2010.

58. WRT Widgets for the S60 on Symbian OS. (WWW-dokumentti.) Forum Nokia. <http://www.forum.nokia.com/document/Widgets_for_the_S60_Platform/>. 2009. Luettu 20.1.2010.

59. Here I am! (WWW-dokumentti.) Aplix corporation. <http://bondidev.omtp.org/widget-gallery/Lists/Widgets/Attachments/7/geo.wgt> 2009. Luettu 29.11.2009.